ФЕДЕРАЛЬНОЕ АГЕНТСТВО ПО ОБРАЗОВАНИЮ
Государственное образовательное учреждение
высшего профессионального образования
ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

И. А. Казакова

Основы языка Transact SQL

Учебное пособие

ПЕНЗА 2010

ФЕДЕРАЛЬНОЕ АГЕНТСТВО ПО ОБРАЗОВАНИЮ
Государственное образовательное учреждение
высшего профессионального образования
«Пензенский государственный университет» (ПГУ)

И. А. Казакова

Основы языка Transact SQL

Рекомендовано Государственным образовательным учреждением высшего профессионального образования «Московский государственный технический университет им. Н. Э. Баумана» в качестве учебного пособия для студентов высших учебных заведений, обучающихся по направлению подготовки 230100 «Информатика и вычислительная техника»

Пенза
Издательство ПГУ
2010

УДК 681.3
К14

Казакова, И. А.
К14 Основы языка Transact SQL : учеб. пособие / И. А. Казакова. – Пенза : Изд-во ПГУ, 2010. – 164 с.

ISBN 978-5-94170-325-8

Рассмотрен язык Transact SQL – диалект языка SQL, используемый в одной из самых популярных систем управления реляционными базами данных – SQL Server 2005. Приведены сведения о структурных подразделах языка – операторах определения данных, операторах манипулирования данными, о языке запросов (представленном оператором SELECT), о средствах управления транзакциями и средствах администрирования базы данных. Все операторы языка подробно проиллюстрированы примерами. Для некоторых примеров приведены возможные результаты.

Учебное пособие подготовлено на кафедре «Математическое обеспечение и применение ЭВМ» и предназначено для студентов, обучающихся по направлению 230100 «Информатика и вычислительная техника», а также для студентов других специальностей, изучающих современные системы управления базами данных и язык Transact SQL.

УДК 681.3

ISBN 978-5-94170-325-8

Введение

SQL (Structured Query Language) – ***Структурированный Язык Запросов*** – стандартный язык запросов по работе с реляционными базами данных. Язык был предложен компанией IBM в начале 1970-х гг. для проверки возможностей реляционной модели.

SQL в чистом (базовом) виде является информационно-логическим языком, а не языком программирования. Однако стандарт языка спецификацией SQL/PSM предусматривает возможность его процедурных расширений, с учетом которых язык уже может рассматриваться в качестве языка программирования.

Первый вариант языка SQL был разработан и частично реализован в рамках проекта экспериментальной реляционной СУБД ***SystemR.*** Исходным названием языка было ***SEQUEL*** – ***Structured English Query Language*** – структурированный английский язык запросов.

До появления SQL в СУБД (независимо от того, на какой модели они основывались) приходилось поддерживать, по крайней мере, 3 языка:

1. ***ЯОД*** – язык определения данных (DDL – Data Definition Language), служащий для создания базы данных (таблиц, индексов и т.д.) и редактирования схемы БД.

2. ***ЯМД*** – язык манипулирования данными (DML – Data Manirulation Language), содержащий оператор для внесения изменений в содержимое таблиц базы данных.

3. ***ЯУД*** **–** язык управления данными (***DCL*** – Data Control Language), содержащий операторы для разграничения доступа пользователей к объектам базы данных.

Язык SQL позволяет решать все эти задачи. SQL предоставляет пользователю достаточно простой и понятный механизм доступа к данным, не связанный с конструированием алгоритма и его описанием на языке программирования высокого уровня. Так, вместо указания того, ***как*** необходимо действовать, пользователь с помо-

щью операторов SQL объясняет СУБД, ***что*** нужно сделать. Далее СУБД сама анализирует текст запроса и определяет, как именно его выполнять (таблица).

История версий стандарта

Год	Название	Изменения
1986	SQL-86	Первый официальный стандарт, принятый институтом ***ANSI*** (*American National Standards Institute*) и одобренный ***ISO*** (*International Organization for Standardization*) в 1987 г. и уточнен в 1989 г.
1992	SQL-92 (SQL2)	Значительные изменения (ISO 9075)
1999	SQL:1999 (SQL3)	Добавлена поддержка регулярных выражений, рекурсивных запросов, поддержка триггеров, базовые процедурные расширения, нескалярные типы данных и некоторые объектно-ориентированные возможности
2003	SQL:2003	Введены расширения для работы с XML-данными, оконные функции (применяемые для работы с OLAP-базами данных, генераторы последовательностей и основанные на них типы данных)
2006	SQL:2006	Функциональность работы с XML-данными значительно расширена
2008	SQL:2008	Улучшены возможности оконных функций, устранены некоторые неоднозначности стандарта SQL:2003

В архитектуре «клиент-сервер» язык SQL занимает очень важное место. Именно он используется как язык общения клиентского программного обеспечения с серверной СУБД, расположенной на удаленном компьютере. Так, клиент посылает серверу запрос на языке SQL, а сервер разбирает его, интерпретирует, выбирает план выполнения, выполняет запрос и отсылает клиенту результат.

Разработан стандарт языка SQL, который является совместной разработкой ***ANSI*** (*American National Standards Institute*) и ***ISO*** (*International Organization for Standardization*).

Достоинства SQL:

1. Наличие международных стандартов.

2. Независимость от конкретной СУБД. Несмотря на наличие диалектов и различий в синтаксисе, в большинстве своем тексты SQL-запросов, содержащие DDL и DML, могут быть достаточно легко перенесены из одной СУБД в другую.

3. Поддержка архитектуры клиент-сервер.

4. Распространенность.

5. Быстрое обучение.

6. Декларативность. С помощью SQL программист описывает только то, какие данные нужно извлечь или модифицировать. Каким образом это сделать, решает СУБД непосредственно при обработке SQL-запроса. Однако программисту полезно представлять, как СУБД будет разбирать текст его запроса. Чем сложнее сконструирован запрос, тем больше он допускает вариантов написания, различных по скорости выполнения, но одинаковых по итоговому набору данных.

Недостатки SQL:

1. Неполное соответствие реляционной модели данных (наличие дубликатов, необязательность первичного ключа, возможность упорядочения результатов).

2. Недостаточно продуманный механизм неопределенных значений.

3. Сложность формулировок и громоздкость.

1. Структура языка Transact SQL

Несмотря на наличие стандартов, практически в каждой СУБД применяется свой диалект языка. Для Microsoft SQL Server 2005 таким языком является Transact SQL. В нем можно выделить следующие подразделы (табл. 1–5).

Таблица 1

Команды языка определения данных (*DDL* – Data Definition Language)

Команда	Смысл	Действие
CREATE TABLE	Создать таблицу	Создает новую таблицу в БД
DROP TABLE	Удалить таблицу	Удаляет таблицу из БД
TRUNCATE TABLE	Удалить данные из таблицы	Уничтожает данные в таблице, но сохраняет ее структуру и индексы
ALTER TABLE	Изменить таблицу	Изменяет структуру существующей таблицы или ограничения целостности, задаваемые для данной таблицы
CREATE VIEW	Создать представление	Создает виртуальную таблицу, соответствующую некоторому SQL-запросу
ALTER VIEW	Изменить представление	Изменяет ранее созданное представление
DROP VIEW	Удалить представление	Удаляет ранее созданное представление
CREATE INDEX	Создать индекс	Создает индекс для некоторой таблицы для обеспечения быстрого доступа по атрибутам, входящим в индекс
DROP INDEX	Удалить индекс	Удаляет ранее созданный индекс

Таблица 2

Команды языка манипулирования данными (*DML* – Data Manipulation Language)

Команда	Смысл	Действие
1	2	3
DELETE	Удалить строки	Удаляет одну или несколько строк, соответствующих условиям фильтрации, из базовой таблицы. Применение команды согласуется с принципами поддержки целостности, поэтому эта команда не всегда может быть выполнена корректно, даже если синтаксически она записана правильно

Окончание табл. 2

1	2	3
INSERT	Вставить строку	Вставляет одну строку в базовую таблицу. Допустимы модификации команды, при которых сразу несколько строк могут быть перенесены из одной таблицы или запроса в базовую таблицу
UPDATE	Обновить строку	Обновляет значения одного или нескольких столбцов в одной или нескольких строках, соответствующих условиям фильтрации

Таблица 3

Язык запросов (*DQL* – Data Query Language)

Команда	Смысл	Действие
SELECT	Выбрать строки	Команда, заменяющая все операторы реляционной алгебры и позволяющая сформировать результирующее отношение, соответствующее запросу

Таблица 4

Средства управления транзакциями

Команда	Смысл	Действие
COMMIT	Завершить транзакцию	Завершить комплексную взаимосвязанную обработку информации, объединенную в транзакцию
ROLLBACK	Откатить транзакцию	Отменить изменения, проведенные в ходе выполнения транзакции
SAVEPOINT	Сохранить промежуточную точку выполнения транзакции	Сохранить промежуточное состояние БД, пометить его для того, чтобы можно было в дальнейшем к нему вернуться

Таблица 5

Средства администрирования БД

Команда	Смысл	Действие
1	2	3
А) Управление БД		
ALTER DATABASE	Изменить БД	Изменить набор основных объектов в базе данных, ограничений, касающихся всей базы данных
CREATE DATABASE	Создать БД	Создать новую базу данных, определив основные параметры для нее

Окончание табл. 5

1	2	3
DROP DATABASE	Удалить БД	Удалить существующую базу данных (только в том случае, если вы имеете право выполнить это действие)
Б) Управление доступом		
GRANT	Предоставить права	Предоставить права доступа на ряд действий над некоторым объектом БД
REVOKE	Лишить прав	Лишить прав доступа к некоторому объекту или некоторым действиям над объектом
DENY	Запретить доступ	Запретить доступ к объектам базы данных

1.1. Идентификаторы

Идентификаторы (имена) в Transact SQL должны состоять из символов латинского алфавита, цифр или символов _ , @ , #. Дополнительно существуют следующие правила:

– идентификаторы должны начинаться с символа латинского алфавита;

– идентификаторы, начинающиеся с @, обозначают локальные переменные;

– идентификаторы, начинающиеся с #, считаются именами временных объектов;

– в имени объектов не могут встречаться пробелы.

Любую колонку или таблицу можно уникально идентифицировать следующим составным именем – имя БД **.** имя владельца **.** имя таблицы или представления, для столбца – названием таблицы или представления. Каждая из этих характеристик отделяется от предыдущей точкой:

database.dbowner.table_name.column_name

database.dbowner.view_name.column_name;

Промежуточные значения – имя владельца может быть опущено, если это не приводит к конфликтам имен.

Если пользователь указывает имя объекта не целиком, то сервер сначала пытается найти его среди объектов, которыми владеет

этот пользователь, после этого производится попытка найти указанный объект как ***database.dbowner.name***.

В процессе работы пользователь может поменять базу данных, к которой он обращается по умолчанию, на любую другую, к которой у него имеется право доступа. Если пользователь не указал явно такую базу данных, то он присоединяется к базе данных ***master*** (основная база данных).

В любом случае, чтобы иметь уверенность, что используется требуемая база данных, нужно использовать команду:

use имя_базы_данных

1.2. Комментарии

В языке Transact SQL возможны две формы записи комментариев:

1. ***/*Текст комментария*/*** – обычно используется для записи многострочных комментариев.

2. ***--Текст комментария*** – используется для комментариев, записываемых в одну строку.

1.3. BNF-нотация

При описании синтаксиса команд Transact SQL обычно используются условные обозначения, известные как ***стандартные формы Бэкуса–Наура*** (***BNF***).

В BNF обозначениях используются следующие элементы:

– Символ "**::=**" означает равенство по определению. Слева от знака стоит определяемое понятие, справа – собственно определение понятия.

– Ключевые слова записываются прописными буквами. Они зарезервированы и составляют часть команды.

– Метки-заполнители конкретных значений элементов и переменных записываются курсивом.

– Необязательные элементы команды заключены в квадратные скобки **[ ]**.

– Вертикальная черта | указывает на то, что все предшествующие ей элементы списка являются необязательными и могут быть заменены любым другим элементом списка после этой черты.

– Фигурные скобки **{ }** указывают на то, что все находящееся внутри них является единым целым.

– Троеточие "**…**" означает, что предшествующая часть команды может быть повторена любое количество раз.

– Многоточие, внутри которого находится запятая "**.,..**", указывает, что предшествующая часть команды, состоящая из нескольких элементов, разделенных запятыми, может иметь произвольное число повторений. Запятую нельзя ставить после последнего элемента.

Замечание. Данное соглашение не входит в стандарт BNF, но позволяет более точно описать синтаксис команд Transact SQL.

– Круглые скобки являются элементом команды.

2. Операторы

Оператор – это символ, обозначающий действие, выполняемое над одним или несколькими выражениями. Чаще всего операторы используются в командах ***DELETE, INSERT, SELECT*** и ***UPDATE***, а также применяются при создании хранимых процедур, функций, триггеров и представлений.

Операторы делятся на следующие категории:

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

2.1. Арифметические операторы

В табл. 6 приведены арифметические операторы.

Таблица 6

Арифметические операторы

Арифметический оператор	Действие
+	Сложение
–	Вычитание
*	Умножение
/	Деление
%	Остаток от деления. Возвращает остаток от деления в виде целого числа

2.2. Операторы присваивания

Оператор присваивания (=) присваивает значение переменной. Ключевое слово ***AS*** служит оператором для присваивания псевдонимов (alias) таблицам или заголовкам столбцов.

2.3. Побитовые операторы

Побитовые операторы являются удобным средством манипулирования битами в двух выражениях целого типа. Для побитовых операций доступны следующие типы данных: ***Binary, Bit, Int, Small Int, Tinyint*** и ***Varbinary***.

В табл. 7 приведены побитовые операторы.

Таблица 7

Побитовые операторы

Побитовые операторы	Действие
&	Побитное И
|	Побитное ИЛИ
~	Побитное НЕ
^	Побитное исключающее ИЛИ

2.4. Операторы сравнения

Операторы сравнения проверяют равенство или неравенство двух выражений. Результатом операции является булево значение – ***TRUE*** или ***FALSE***.

Замечание. По стандарту ANSI при сравнении выражений, если хотя бы одно из них равно ***NULL***, результатом будет ***NULL***.

Например, выражение ***15 + NULL*** в результате выдает ***NULL.*** Результатом выражения ***Oct 10, 2010 + NULL*** также будет ***NULL.***

В табл. 8 приведены операторы сравнения.

Таблица 8

Операторы сравнения

Действие	Оператор сравнения
=	Равно
>	Больше
<	Меньше
>=	Больше или равно
<=	Меньше или равно
<>	Не равно

Операторы сравнения чаще всего используются в предложениях ***WHERE*** для отбора строк, соответствующих условиям поиска.

Например, можно выбрать студентов, получающих стипендию больше тысячи рублей.

SELECT FIO FROM Students

WHERE Stipendiya >=1000;

2.5. Логические операторы

Логические операторы обычно применяются в предложении ***WHERE*** для проверки истинности какого-либо условия. Логические операторы возвращают булево значение ***TRUE*** или ***FALSE.***

В табл. 9 приведены операторы сравнения.

Таблица 9

Операторы сравнения

Логический оператор	Действие
ALL	***TRUE***, если весь набор сравнений дает результат ***TRUE***
AND	***TRUE***, если оба булевых выражения дают результат ***TRUE***
ANY	***TRUE***, если хотя бы одно сравнение из набора дает 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	Обращает значение любого другого булева оператора
OR	***TRUE***, если любое булево выражение равно ***TRUE***
SOME	***TRUE***, если несколько сравнений из набора дают результат ***TRUE***

2.6. Унарные операторы

Унарные операторы выполняют операцию над одним выражением любого типа, относящимся к числовой категории.

В табл. 10 приведены унарные операторы.

Таблица 10

Унарные операторы

Унарный оператор	Действие
+	Числовое значение становится положительным
-	Числовое значение становится отрицательным
~	Поразрядное НЕ. Возвращает двоичное дополнение числа

2.7. Приоритет операторов

Если в выражении присутствует несколько операторов, то порядок их выполнения определяется приоритетом операторов. Ниже перечислены уровни приоритета операторов (от самого высокого к самому низкому).

1. () – выражения в скобках.

2. +, -, ~ – унарные операторы.

3. *, /, % – арифметические операторы типа умножения.

4. +, - – арифметические операторы типа сложения.

5. =, >, <, >=, <=, <> – операторы сравнения.

6. ^ (побитное исключающее ИЛИ), & (побитное И), | (побитное ИЛИ).

7. ***NOT***.

8. ***AND***.

9. ***ALL, ANY, BETWEEN, IN, LIKE, OR, SOME***.

10. = – присваивание значения переменной.

Если операторы имеют одинаковый приоритет, вычисления производятся слева направо. Для изменения порядка выполнения операторов используются скобки. Выражения в скобках вычисляются первыми.

Пример. Вычисляется сумма 2 + 2 · 5 с результатом 12:

SELECT 2+2*5
FROM Teachers

Пример 2. Вычисляется сумма (2 + 2) · 5 с результатом 20:

SELECT (2+2)*5
FROM Teachers

Если применяются вложенные скобки, то первыми вычисляются выражения в наиболее глубоко вложенных скобках.

Краткие итоги

В языке **Transact SQL** можно выделить следующие подразделы:

1. Команды языка определения данных (***DDL*** – Data Definition Language).

2. Команды языка манипулирования данными (***DML*** – Data Manipulation Language).

3. Язык запросов (***DQL*** – Data Query Language).

4. Средства управления транзакциями.

5. Средства администрирования БД.

Идентификаторы (имена) в Transact SQL должны состоять из символов латинского алфавита, цифр или из символов _ , @ , #.

В языке Transact SQL возможны две формы записи комментариев:

1. ***/*Текст комментария*/*** – обычно используется для записи многострочных комментариев.

2. ***--Текст комментария*** – используется для комментариев, записываемых в одну строку.

Оператор – это символ, обозначающий действие, выполняемое над одним или несколькими выражениями. Операторы делятся на следующие категории:

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

Контрольные вопросы

1. Какие языки баз данных объединены в языке Transact SQL?
2. Как записываются комментарии в языке Transact SQL?
3. Какие классы операторов существуют в языке Transact SQL?

3. Типы данных

Один из основных моментов процесса создания таблицы – определение типов данных для ее полей. Тип данных поля таблицы определяет тип информации, которая будет размещаться в этом поле. Понятие типа данных в SQL Server 2005 (табл. 11) полностью адекватно понятию типа данных в современных языках программирования.

Таблица 11

Типы данных

Тип данных		Занимаемая память (в байтах)	Точ-ность	Внут-ренняя точность (используется сервером)	Внут-ренний масштаб (используется сервером)
1	2	3	4	5	6
Двоичные	Binary	8000	8000	0	0
	Varbinary	8000	8000	0	0
	Bit	1	1	1	0
Символьные	Char	8000	8000	0	0
	Varchar	8000	8000	0	0
Символьные (Unicode)	Nchar	8000	4000	0	0
	Nvarchar	8000	4000	0	0
Дата и время	Datetime	8	23	23	3
	Smalldatetime	4	16	16	1
Точное представление чисел	Decimal	17	38	38	38
	Numeric	17	38	38	
	Bigint	8	19	19	0
Представление чисел с плавающей точкой	Float	8	53	53	0
	Real	4	24	24	0
Текст и картинки	Text	16	Null	0	0
	Ntext (Unicode)	16	Null	0	0
	Image	16	Null	0	0
Целочисленные типы	Int	4	10	10	0
	Smallint	2	5	5	0
	Tinyint	1	3	3	0
Денежные типы	Money	8	19	19	4
	Smallmoney	4	10	10	4

Окончание табл. 11

1	2	3	4	5	6
Специальные	Timestamp	8	8	0	0
	Sql_variant	8016	0	0	0
	Sysname	256	128	0	0
	Uniqueidentifier	16	16	0	0
	Типы, определяемые пользователем				

3.1. Двоичные типы

BIT позволяет хранить один бит, который принимает значения ***0, 1*** или ***NULL***. При вводе числа, отличного от ***1***, принимается значение, равное ***1***. Тип данных ***bit*** имеет размер в один байт, но при наличии нескольких полей типа ***bit*** в таблице они все будут упакованы вместе. Например, если у нас есть семь полей типа ***bit***, то суммарное занимаемое ими пространство будет равно одному байту.

BINARY [длина] – хранит двоичное значение фиксированной длины от 1 до 8 000 байт. Значение типа ***BINARY*** занимает **(*длина + 4*) *байта***.

VARBINARY [длина] – представляет собой двоичное значение переменной длины до 8000 байт. Занимаемое место равно размеру вставленных данных плюс 4 байта.

3.2. Символьные типы данных

CHAR (длина) – хранит символьные данные фиксированной длины от 1 до 8000 символов.

VARCHAR (длина) – строка имеет переменную длину.

NCHAR (длина).

NVARCHAR (длина).

Последние два типа предназначены для хранения символов ***Unicode***. Максимальное значение длины ограничено 8000 знаками.

Параметр ***длина*** применяется для указания максимального количества символов, которые могут быть помещены в данный столбец (по умолчанию ***1***).

Если строка определена с ***фиксированной*** длиной значений, то при вводе в нее меньшего количества символов значение дополняется до указанной длины пробелами, добавляемыми справа.

Если строка определена с ***переменной*** длиной значений, то при вводе в нее меньшего количества символов в БД будут сохранены только введенные символы, что позволит достичь определенной экономии внешней памяти.

Хранение символьных данных большого объема (до 2 Гб) осуществляется с помощью текстовых типов данных ***TEXT*** (16 байт) и ***NTEXT*** (16 байт).

3.3. Числовые типы

1. ***Целочисленные***.

INT (INTEGER) – этот тип используется для представления целых чисел со знаком или без знака в диапазоне от -2^{31} до $2^{31} - 1$. Для этих данных в памяти компьютера отводится 4 байта. Все целочисленные типы, а также типы, хранящие десятичные дроби, поддерживают свойство ***IDENTITY***. ***IDENTITY*** – это автоматически инкрементируемый идентификатор строки.

SMALLINT – хранит целые числа со знаком или без знака в диапазоне от -2^{15} до $2^{15} - 1$. Для этих данных в памяти компьютера отводится 2 байта.

TINYINT – этот тип используется для представления данных в диапазоне от 0 до 255. Для этих данных в памяти компьютера отводится 1 байт.

BIGINT – этот тип используется для представления данных в диапазоне от -2^{63} до $2^{63} - 1$. Для этих данных в памяти компьютера отводится 8 байт. В типе ***INT*** указаны особенности свойства ***IDENTITY,*** также применимые к типу ***BIGINT***.

2. ***Нецелочисленные***, т.е. числа, в составе которых есть десятичная точка. Нецелочисленные данные разделяются на два типа:

а) ***десятичные***.

К десятичным типам данных относятся *типы* ***DECIMAL*** или ***DEC*** и ***NUMERIC***. Эти типы данных позволяют самостоятельно определить формат точности числа с плавающей запятой.

DECIMAL [(точность[,масштаб])] или ***DEC –*** хранит десятичные дроби длиной до 38 цифр. К типу ***DECIMAL также применимо свойство IDENTITY.***

NUMERIC [(точность[,масштаб])] – синоним типа ***DECIMAL.***

Параметр ***точность*** указывает максимальное количество цифр вводимых данных этого типа (до и после десятичной точки в сумме, без учета самой десятичной точки).

Параметр ***масштаб*** указывает максимальное количество цифр, расположенных после десятичной точки. В обычном режиме сервер позволяет вводить не более 28 цифр, используемых в типах ***DECIMAL*** и ***NUMERIC*** (от 2 до 17 байт). По умолчанию длина дробной части равна нулю;

б) ***приблизительные (округленные) числа***.

Приблизительные числа или числа с плавающей точкой представляются в научной нотации, при которой число записывается с помощью мантиссы, умноженной на определенную степень десяти (порядок), например: 10E3, +5.2E6, –0.2E – 4. Это обеспечивает одинаковую точность вычислений независимо от того, насколько мало или велико значение.

Для определения данных вещественного типа используется формат:

<вещественный_тип>::=

***{ FLOAT [точность] | REAL }*;**

Параметр ***точность*** задает количество значащих цифр мантиссы.

Тип ***FLOAT*** используется для представления данных, содержащих до 15 цифр. Хранит значения с плавающей точкой в диапазоне от –1.79E + 308 до 1.79E + 308. Для этих данных в памяти компьютера отводится 8 байт.

Тип ***REAL*** – хранит значения с плавающей точкой в диапазоне от –3.40E + 38 до 3.40E + 38. Используется для представления данных, содержащих до 7 цифр. Для этих данных в памяти компьютера отводится 4 байта.

3.4. Дата и время

Дата и время представляются алфавитно-цифровыми данными в виде строки.

По умолчанию для отображения даты используется ***формат Mon dd yyyy hh:mmAM***, например, ‘Apr 10 2010 10:23AM’.

При вводе данных следует заключать значение в одиночные кавычки.

DATETIME

Этот тип данных имеет размер в 8 байт, т.е. два четырехбайтных целых – 4 байта на количество дней, прошедших или еще не наступивших с 1 января 1900, и 4 байта на число миллисекунд, прошедших с полуночи.

DATETIME может содержать значение даты и времени с 1 января 1753 г. 00:00:00 по 31 декабря 9999 г. 23:59:59, с точностью в три тысячных секунды. По умолчанию datetime имеет значение 1 января 1900 г., полдень.

Замечание. Даты ранее 1 января 1753 г. не поддерживаются, так как англоговорящий мир именно 1 января 1753 г. перешел на григорианский календарь, а преобразование дат юлианского календаря в григорианский довольно сложно.

SMALLDATETIME

Тип данных, во многом аналогичный datetime, но менее точный. Размер его – 4 байта, 2 байта на число дней, прошедших с 1 января 1900 г., и 2 байта на число минут с полуночи. Даты могут быть представлены в диапазоне с 1 января 1900 г. по 6 июня 2079 г., с точностью в минуту.

3.5. Денежный тип

Делает возможным хранение информации денежного типа. Этот тип обеспечивает точность значений до четырех знаков после запятой.

MONEY – хранит денежные значения в диапазоне от –922 337 203 685 477.5808 до +922 337 203 685 477.5807. Значение занимает 8 байт.

SMALLMONEY – хранит денежные значения в диапазоне от –214 748.3648 до +214 748.3647, размер – 4 байта.

3.6. Тип IMAGE

IMAGE хранит двоичное значение переменной длины до 2 147 483 647 байт. Этот тип данных используется для хранения графики, звука и документов, например, Microsoft Word, Microsoft Excel. Данные типа ***IMAGE*** имеют множество ограничений на способы использования.

3.7. Специальные типы

В среде SQL Server реализован ряд специальных типов данных.

TIMESTAMP – хранит автоматически генерируемое двоичное число, обеспечивающее уникальность в текущей базе данных. Поля типа ***TIMESTAMP*** не имеют отношения к системной дате или системному времени ***TIMESTAMP*** имеет размер 8 байт и представлено как ***VARBINARY(8);***

UNIQUEIDENTIFIER – используется для хранения глобальных уникальных идентификационных номеров. Это значения, уникальные для всех баз данных и всех серверов; представлено в виде

XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX, где каждый ***X*** представляет собой шестнадцатеричное число в диапазоне 0–9 или A–F. Единственные операции, которые можно выполнять над значениями этого типа, являются сравнение и проверка на ***NULL***;

SYSNAME –предназначен для идентификаторов объектов;

SQL_VARIANT (n) **–** позволяет хранить значения любого из поддерживаемых SQL Server типов данных за исключением ***TEXT***, ***NTEXT***, ***IMAGE*** и ***TIMESTAMP***.

3.8. Получение информации о типах данных

Получить список всех типов данных, включая пользовательские, можно из системной таблицы ***systypes***:

SELECT * FROM systypes

3.9. Преобразование типов

Наиболее часто выполняется конвертирование чисел в символьные данные и наоборот. Для этого используется специальная функция ***STR***.

Для выполнения других преобразований SQL Server предлагает универсальные функции ***CONVERT*** и ***CAST***, с помощью которых значения одного типа преобразовываются в значения другого типа (если такие изменения возможны). ***CONVERT*** и ***CAST*** примерно одинаковы и могут быть взаимозаменяемыми.

CONVERT (тип_данных[(длина)], выражение [,стиль])

CAST (выражение AS тип_данных)

С помощью аргумента ***стиль*** можно управлять стилем представления значений следующих типов данных: дата/время, денежный или нецелочисленный.

Пример 3. Использование функции ***Cast.***

INSERT INTO Teachers (FIO, Data_Rozhd,Adres, Stazh)

VALUES (Николаева Нина Валерьевна', cast('1977.01.07' AS Datetime), 'ул. Лермонтова, д.7 кв. 16', 14)

Результат выполнения оператора приведен на рис. 1.

Results | Messages

	ID_Teacher	FIO	Data_Rozhd	Adres	Stazh
1	3	Николаева Нина Валерьевна	1977-01-07 00:00:00.000	ул. Лермонтова, д.7 кв. 16	14

Рис. 1. Использование функции *Cast*

Пример 4. Использование функции ***Conert***

SELECT FIO AS ФИО, CONVERT (Varchar(25), Data_Rozhd,5) AS Дата_Рождения FROM Teachers

Результат выполнения оператора приведен на рис. 2.

Results | Messages

	ФИО	Дата_Рождения
1	Николаева Нина Валерьевна	07-01-77

Рис. 2. Использование функции *Conert*

Краткие итоги

Тип данных поля таблицы определяет тип информации, которая будет размещаться в этом поле. Понятие типа данных в SQL Server 2005 полностью адекватно понятию типа данных в современных языках программирования.

В SQL Server 2005 используются следующие типы данных:

1. Двоичные типы (***BIT, BINARY, VARBINARY***).

2. Символьные типы данных (***CHAR***, ***VARCHAR***, ***NCHAR***, ***NVARCHAR, TEXT***).

3. Числовые типы:

А) Целочисленные (***INTEGER, SMALLINT, TINYINT, BIGINT***).

Б) Нецелочисленные:

– Десятичные ***(DECIMAL, NUMERIC)***

– Приблизительные (округленные) (***FLOAT*** | ***REAL***)

4. Дата и время (***DATETIME, SMALLDATETIME***).

5. Денежный тип **(*MONEY, SMALLMONEY***).

6. Тип ***IMAGE*** – используется для хранения графики, звука и документов.

7. Специальные типы (***TIMESTAMP***, ***UNIQUEIDENTIFIER***, ***SYSNAME***, ***SQL_VARIANT***).

Существуют функции преобразования типов – ***STR***, ***CONVERT*** и ***CAST***, с помощью которых значения одного типа преобразовываются в значения другого типа.

Контрольные вопросы

1. Что такое тип данных в контексте баз данных?

2. Что определяет тип данных поля таблицы?

3. К какому типу относятся ***DECIMAL, NUMERIC***?

4. В чем различие между типами ***CHAR*** и ***VARCHAR***?

4. Встроенные функции

Основные встроенные функции, имеющиеся в распоряжении пользователей при работе с SQL:

- математические функции;
- строковые функции;
- функции для работы с датой и временем.

4.1. Математические функции

Краткий обзор математических функций представлен в табл. 12.

Таблица 12

Математические функции

Функция	Действие
ABS	Вычисляет абсолютное значение числа
ACOS	Вычисляет арккосинус
ASIN	Вычисляет арксинус
ATAN	Вычисляет арктангенс
ATN2	Вычисляет арктангенс с учетом квадратов
CEILING	Выполняет округление вверх
COS	Вычисляет косинус угла
COT	Возвращает котангенс угла
DEGREES	Преобразует значение угла из радиан в градусы
EXP	Возвращает экспоненту
FLOOR	Выполняет округление вниз
LOG	Вычисляет натуральный логарифм
LOG10	Вычисляет десятичный логарифм
PI	Возвращает значение «пи»
POWER	Возводит число в степень
RADIANS	Преобразует значение угла из градуса в радианы
RAND	Возвращает случайное число
ROUND	Выполняет округление с заданной точностью
SIGN	Определяет знак числа
SIN	Вычисляет синус угла
SQUARE	Выполняет возведение числа в квадрат
SQRT	Извлекает квадратный корень
TAN	Возвращает тангенс угла

Пример 5. Использование функции округления до одного знака после запятой. Увеличить размер стипендии на 10 %.

SELECT Fio AS ФИО, Stipendiya AS Старая_стипендия, (Stipendiya+ROUND(Stipendiya*0.1,1)) AS Новая_стипендия

FROM Students

WHERE Stipendiya IS NOT NULL

Результат выполнения запроса приведен на рис. 3.

Results | Messages

	ФИО	Старая_стипендия	Новая_стипендия
1	Макарь В.А.	800	880.0
2	Ивкин И.Ю.	1700	1870.0
3	Заваровский К.Ю.	800	880.0
4	Заваровский К.В.	1200	1320.0
5	Иванчуков А.Г.	800	880.0
6	Горин А.А.	1700	1870.0
7	Коряшкин А.С.	1700	1870.0
8	Янин А.В.	1200	1320.0

Рис. 3. Использование функции округления

4.2. Строковые функции

Краткий обзор функций представлен в табл. 13.

Таблица 13

Строковые функции

Функция	Действие
1	2
ASCII	Возвращает код ASCII левого символа строки
CHAR	По коду ASCII возвращает символ
CHARINDEX	Определяет порядковый номер символа, с которого начинается вхождение подстроки в строку
DIFFERENCE	Возвращает показатель совпадения строк
LEFT	Возвращает указанное число символов с начала строки
LEN	Возвращает длину строки
LOWER	Переводит все символы строки в нижний регистр
LTRIM	Удаляет пробелы в начале строки
NCHAR	Возвращает по коду символ Unicode
PATINDEX	Выполняет поиск подстроки в строке по указанному шаблону
REPLACE	Заменяет вхождения подстроки на указанное значение
QUOTENAME	Конвертирует строку в формат Unicode
REPLICATE	Выполняет тиражирование строки определенное число раз
REVERSE	Возвращает строку, символы которой записаны в обратном порядке

Окончание табл. 13

1	2
RIGHT	Возвращает указанное число символов с конца строки
RTRIM	Удаляет пробелы в конце строки
SOUNDEX	Возвращает код звучания строки
SPACE	Возвращает указанное число пробелов
STR	Выполняет конвертирование значения числового типа в символьный формат
STUFF	Удаляет указанное число символов, заменяя новой подстрокой
SUBSTRING	Возвращает для строки подстроку указанной длины с заданного символа
UNICODE	Возвращает Unicode-код левого символа строки
UPPER	Переводит все символы строки в верхний регистр

Пример 6. Использование функции ***LEFT*** для получения инициалов преподавателей.

SELECT Familia + ’ ’ + LEFT (Imja,1)+’.’ + LEFT (Surname,1) +’.’ AS ФИО

FROM Teachers;

Результат выполнения запроса приведен на рис. 4.

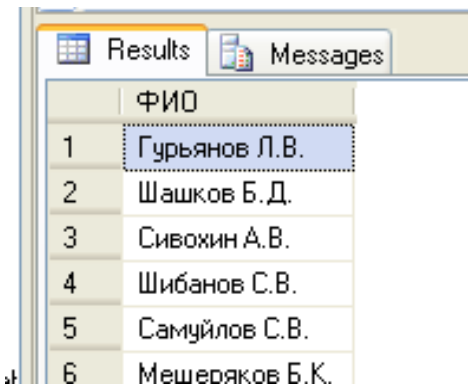


Рис. 4. Использование функции *LEFT*

4.3. Функции для работы с датой и временем

Основные функции для работы с датой и временем представлены в табл. 14.

Таблица 14

Функции для работы с датой и временем

Функция	Действие
DATEADD	Добавляет к дате указанное значение дней, месяцев, часов и т.д.
DATEDIFF	Возвращает разницу между указанными частями двух дат
DATENAME	Выделяет из даты указанную часть и возвращает ее в символьном формате
DATEPART	Выделяет из даты указанную часть и возвращает ее в числовом формате
DAY	Возвращает число из указанной даты
GETDATE	Возвращает текущее системное время
ISDATE	Проверяет правильность выражения на соответствие одному из возможных форматов ввода даты
MONTH	Возвращает значение месяца из указанной даты
YEAR	Возвращает значение года из указанной даты

Пример 7. Использование функций ***YEAR*** и ***MONTH*** для определения месяца и года по дате рождения.

SELECT YEAR(Data_Rozhd) AS Год, MONTH(Data_Rozhd)AS Месяц

FROM Teachers;

Краткие итоги

Основные встроенные функции, имеющиеся в распоряжении пользователей при работе с TRANSACT SQL:

– математические функции;

– строковые функции;

– функции для работы с датой и временем.

Контрольные вопросы

1. Какие типы встроенных функций существуют в TRANSACT SQL?

2. К какому типу относятся функции ***ABS***, ***LOG*** и ***SQRT***?

3. К какому типу относятся функции ***RTRIM***, ***STR*** и ***UPPER***?

4. К какому типу относятся функции ***GETDATE***, ***MONTH*** и ***DATEADD***?

5. Основные объекты баз данных SQL Server

Логически данные в SQL Server 2005 организованы в виде объектов. Основные объекты баз данных SQL Server представлены в табл. 15.

Таблица 15

Основные объекты баз данных SQL Server

Объект	Характеристика
1	2
Tables	Таблицы базы данных, в которых хранятся собственно данные Таблицы содержат: – ***строки***; каждая строка (или запись) представляет собой совокупность атрибутов (свойств) конкретного экземпляра *объекта*; – ***столбцы***; каждый столбец (поле) представляет собой атрибут или совокупность атрибутов. Поле строки является минимальным элементом таблицы. Каждый столбец в таблице имеет определенное имя, тип данных и размер
Views	Представления (виртуальные таблицы) для отображения данных из таблиц Подобно реальным таблицам, ***Views*** содержат именованные столбцы и строки с данными. Для конечных пользователей представление выглядит как таблица, но в действительности оно не содержит данных, а лишь представляет данные, расположенные в одной или нескольких таблицах. Информация, которую видит пользователь через представление, не сохраняется в базе данных как самостоятельный объект
Stored Procedures	Хранимые процедуры – это группа команд SQL, объединенных в один модуль. Такая группа команд компилируется и выполняется как единое целое
Triggers	Триггеры – специальный класс хранимых процедур, автоматически запускаемых при добавлении, изменении или удалении данных из таблицы
User Defined function	Создаваемые пользователем функции, т.е. конструкции, содержащие часто исполняемый код. Функция выполняет какие-либо действия над данными и возвращает некоторое значение

Окончание табл. 15

1	2
Indexes	Индекс – структура, связанная с таблицей или представлением и предназначенная для ускорения поиска информации в них. Он содержит отсортированные значения индексированного столбца или столбцов со ссылками на соответствующую строку исходной таблицы или представления. Повышение производительности достигается за счет сортировки данных. Использование индексов может существенно повысить производительность поиска, однако для хранения индексов необходимо дополнительное пространство в базе данных
User Defined Data Types	Определяемые пользователем типы данных – это типы данных, которые создает пользователь на основе системных типов данных. Например, в нескольких таблицах необходимо хранить однотипные значения; причем нужно гарантировать, что столбцы в таблице будут иметь одинаковый размер, тип данных и чувствительность к значениям ***NULL***
Constraints	Ограничение целостности – это объекты для обеспечения логической целостности данных Ограничения целостности – механизм, обеспечивающий автоматический контроль соответствия данных установленным условиям (или ограничениям). Ограничения целостности имеют приоритет над триггерами, правилами и значениями по умолчанию. К ограничениям целостности относятся: ограничение на значение ***NULL***, проверочные ограничения, ограничение уникальности (уникальный ключ), ограничение первичного ключа и ограничение внешнего ключа. Последние три ограничения тесно связаны с понятием ключей
Keys	Ключи – один из видов ограничений целостности данных
Users	Пользователи, обладающие доступом к базе данных
Roles	Роли, позволяющие объединять пользователей в группы
Rules	Правила используются для ограничения значений, хранимых в столбце таблицы или в пользовательском типе данных. Они существуют как самостоятельные объекты базы данных, которые связываются со столбцами таблиц и пользовательскими типами данных. Контроль значений данных может быть реализован и с помощью ограничений целостности
Defaults	Умолчания – объект базы данных, представляющий значение, которое будет присвоено элементу таблицы при вставке строки, если в команде вставки явно не указано значение для этого столбца

6. База данных

6.1. Создание базы данных

В стандарте ANSI нет команды ***CREATE DATABASE.*** Но почти все платформы СУБД поддерживают какой-либо вариант этой команды.

Процедура создания базы данных обычно закрепляется только за администратором базы данных.

***Этапы создания* БД**

1) **создание** базы данных (файл с расширением ****.mdf*** для основных файлов и файл с расширением ****.ndf*** для вторичных файлов). В файле базы данных записываются сведения об основных объектах (таблицах, индексах, просмотрах и т.д.);

2) **создание журнала транзакций,** принадлежащего базе данных (файл с расширением ****.ldf***). Здесь записываются сведения о процессе работы с транзакциями (контроль целостности данных, состояния базы данных до и после выполнения транзакций):

<определение_базы_данных> ::=

CREATE DATABASE имя_базы_данных

[ON [PRIMARY]

[<определение_файла> [,...n]]

[,<определение_группы> [,...n]]]

[LOG ON {<определение_файла>[,...n] }]

имя_базы_данных – стандартный идентификатор, допустимый в SQL. Если имя базы данных содержит пробелы или любые другие недопустимые символы, оно заключается в ограничители (двойные кавычки или квадратные скобки). Имя базы данных должно быть уникальным в пределах сервера и не может превышать 128 символов.

Если в процессе использования базы данных планируется ее размещение на нескольких дисках, то можно создать так называемые ***вторичные*** файлы базы данных с расширением ****.ndf***. В этом случае основная информация о базе данных располагается в ***первичном*** (***PRIMARY***) файле, а при нехватке для него свободного места добавляемая информация будет размещаться во вторичном

файле. Подход, используемый в SQL-сервере, позволяет распределять содержимое базы данных по нескольким дисковым томам.

ON – определяет список файлов на диске для размещения информации, хранящейся в базе данных.

PRIMARY – определяет первичный (основной) файл. В базе данных такой файл может быть только один. Если он опущен, то основным является первый файл в списке. Основной файл содержит логическое начало базы данных.

При создании базы данных можно определить набор файлов, из которых она будет состоять. Файл определяется с помощью следующей конструкции:

<определение_файла>::=

([NAME=логическое_имя_файла,]

FILENAME=’физическое_имя_файла’

[,SIZE=размер_файла]

[,MAXSIZE={max_размер_файла |UNLIMITED }]

[, FILEGROWTH=величина_прироста])[,...n]

NAME=логическое_имя_файла – это имя файла, под которым он будет распознаваться при выполнении различных SQL-команд.

FILENAME=’физическое_имя_файла’ – это имя файла, который будет создан на жестком диске. Это имя останется за файлом на уровне операционной системы.

SIZE=размер_файла определяет первоначальный размер файла; минимальный размер параметра – ***512 Кб;*** если он не указан, то по умолчанию принимается ***1 Мб***.

MAXSIZE={max_размер_файла} определяет максимальный размер файла базы данных. При значении параметра ***UNLIMITED*** максимальный размер базы данных ограничивается свободным местом на диске.

FILEGROWTH=величина_прироста – величина автоматического прироста размера базы данных. Приращение – это либо абсолютная величина в мегабайтах либо процентное соотношение. Если ***FILEGROWTH*** не задан, то файл за одно увеличение будет увеличиваться на 10 % (но не менее, чем на 64 Кб.)

Дополнительные файлы могут быть включены в группу:

<определение_группы>::=FILEGROUP имя_группы_файлов
<определение_файла>[,...n]

LOG ON {<определение_файла>[,...n] } – здесь описываются файл или файлы, в которых хранится журнал транзакций.

Пример 8. Создать базу данных, причем для данных определить три файла на дисках ***D***, ***E, F***, для журнала транзакций – два файла на дисках ***H*** и ***M***:

```
CREATE DATABASE Institute
ON PRIMARY
 (NAME=Archiv1,
 FILENAME="d:\user\data\archdat1.mdf",
 SIZE=100MB, MAXSIZE=200, FILEGROWTH=20),
 (NAME=Archiv2,
 FILENAME="e:\user\data\archdat2.mdf",
 SIZE=100MB, MAXSIZE=200, FILEGROWTH=20),
 (NAME=Archiv3,
 FILENAME="f:\user\data\archdat3.mdf",
 SIZE=100MB, MAXSIZE=200, FILEGROWTH=20)
 LOG ON
 (NAME=Archlog1,
 FILENAME="h:\user\data\archlog1.ldf",
 SIZE=100MB, MAXSIZE=200, FILEGROWTH=20),
 (NAME=Archlog2,
 FILENAME="m:\user\data\archlog2.ldf",
 SIZE=100MB, MAXSIZE=200, FILEGROWTH=20);
```

Краткая форма оператора создания базы данных –

CREATE DATABASE имя_базы_данных;

В этом случае все значения параметров задаются по умолчанию.

Пример 9. Создать базу данных ***Institute*** с параметрами по умолчанию:

CREATE DATABASE Institute;

6.2. Изменение базы данных

Большинство действий по изменению конфигурации базы данных выполняется с помощью следующей команды:

<изменение_базы_данных> ::=

ALTER DATABASE имя_базы_данных

{ ADD FILE <определение_файла>[,...n]

[TO FILEGROUP имя_группы_файлов]

| ADD LOG FILE <определение_файла>[,...n]

| REMOVE FILE логическое_имя_файла

| ADD FILEGROUP имя_группы_файлов

| REMOVE FILEGROUP имя_группы_файлов

| MODIFY NAME = new_database_name

| MODIFY FILEGROUP имя_группы_файлов

<свойства_группы_файлов>};

Как видно из синтаксиса, за один вызов команды может быть изменено не более одного параметра конфигурации базы данных. Если необходимо выполнить несколько изменений, придется разбить процесс на ряд отдельных шагов.

В базу данных можно добавить (***ADD***) новые файлы данных (в указанную группу файлов или в группу, принятую по умолчанию) или файлы журнала транзакций.

Параметры файлов и групп файлов можно изменять (***MODIFY***).

Для удаления из базы данных файлов или групп файлов используется параметр ***REMOVE***. Однако удаление файла возможно лишь при условии его освобождения от данных. В противном случае сервер не разрешит удаление.

В качестве свойств группы файлов используются следующие:

READONLY – группа файлов используется только для чтения;

READWRITE – в группе файлов разрешаются изменения;

DEFAULT – указанная группа файлов принимается по умолчанию.

Пример 10. Переименовать базу данных

ALTER DATABASE Institute MODIFY NAME = Archiv

6.3. Удаление базы данных

Удаление базы данных осуществляется командой

DROP DATABASE имя_базы_данных [,...n];

Удаляются все содержащиеся в базе данных объекты, а также файлы, в которых она размещается. Для исполнения операции удаления базы данных пользователь должен обладать соответствующими правами.

Пример 11. Удалить базу данных ***Institute***

DROP DATABASE Institute;

Краткие итоги

Процедура создания базы данных обычно закрепляется только за администратором базы данных.

Этапы создания базы данных:

1) создание базы данных;

2) создание журнала транзакций.

Создание базы данных выполняется с помощью команды ***CREATE DATABASE*** (с параметрами).

Краткая форма оператора создания базы данных –

CREATE DATABASE имя_базы_данных;

В этом случае все значения параметров задаются по умолчанию.

Изменение базы данных выполняется с помощью команды

ALTER DATABASE имя_базы_данных;

Удаление базы данных осуществляется командой

DROP DATABASE имя_базы_данных [,...n];

Контрольные вопросы

1. За кем закреплена процедура создания базы данных?

2. Какие команды используются:

а) для создания базы данных;

б) изменения базы данных;

в) удаления базы данных?

7. Таблицы

7.1. Создание таблицы

Таблица – основной объект для хранения информации в реляционной базе данных.

В SQL Server 2005 в одной базе данных может быть до 2 миллиардов таблиц. В таблице – 1024 столбца, в 1 строке – 8060 байтов.

После создания общей структуры базы данных можно приступить к созданию таблиц, которые представляют собой отношения, входящие в состав проекта базы данных.

Таблицы базы данных создаются с помощью команды ***CREATE TABLE***. Эта команда создает пустую таблицу, т.е. таблицу, не имеющую строк. Значения в эту таблицу вводятся с помощью команды ***INSERT***. Команда ***CREATE TABLE*** определяет имя таблицы и множество поименованных столбцов в указанном порядке. Для каждого столбца должен быть определен тип и размер.

Тип данных, для которого обязательно должен быть указан размер, – это ***CHAR***. Реальное количество символов, которое может находиться в поле, изменяется от нуля (если в поле содержится NULL–значение) до заданного в ***CREATE TABLE*** максимального значения.

Упрощенный синтаксис этой команды (табл. 16):

CREATE TABLE <имя таблицы>

({<имя поля> <тип данных> [(<размер>)]

[<ограничения целостности поля>…]} .,..

[, <ограничения целостности таблицы>.,..]);

Таблица 16

Описание команды *CREATE TABLE*

Элемент	Описание
<имя таблицы>	[*database*.[*owner*].]*table_name*
<имя столбца >	Имя столбца таблицы, обычный идентификатор
<тип данных>	Тип данных поля
<размер>	Размер поля в символах (для текста и чисел)

Базовый синтаксис команды создания таблицы имеет следующий вид:

<определение_таблицы> ::=

CREATE TABLE имя_таблицы

(

{

имя_столбца тип_данных [NOT NULL] [[PRIMARY KEY | UNIQUE]

[DEFAULT <значение>]

[IDENTITY [(стартовое_значение, инкремент)]]

[FOREIGN KEY

REFERENCES имя_род_таблицы [(имя_столбца_род_таблицы)]

[CHECK (<условие_выбора>)] [,...n]

[ON UPDATE {CASCADE | NO ACTION }]

[ON DELETE {CASCADE | NO ACTION }]

}

);

[IDENTITY [(стартовое_значение, инкремент)] – для колонки с таким свойством сервером автоматически генерируется возрастающая последовательность. Отсчет начинается со стартового значения, которое увеличивается на величину инкремента. Если какой-либо параметр опущен, то по умолчанию принимается единица. Сервер не гарантирует непрерывность значений – в реальных данных в таблице могут появляться разрывы.

Далее в данном пособии будет рассматриваться база данных ***Institute***, имеющая структуру, приведенную на рис. 5.

База данных состоит из 6 таблиц:

Таблица ***Teachers*** содержит сведения о преподавателях;

Таблица ***Lessons*** содержит сведения о предметах;

Таблица ***Groups*** содержит сведения об учебных группах;

Таблица ***Students*** содержит сведения о преподавателях;

Таблица ***Kafedra*** содержит сведения о кафедрах;

Таблица ***Progress*** содержит сведения об успеваемости студентов.

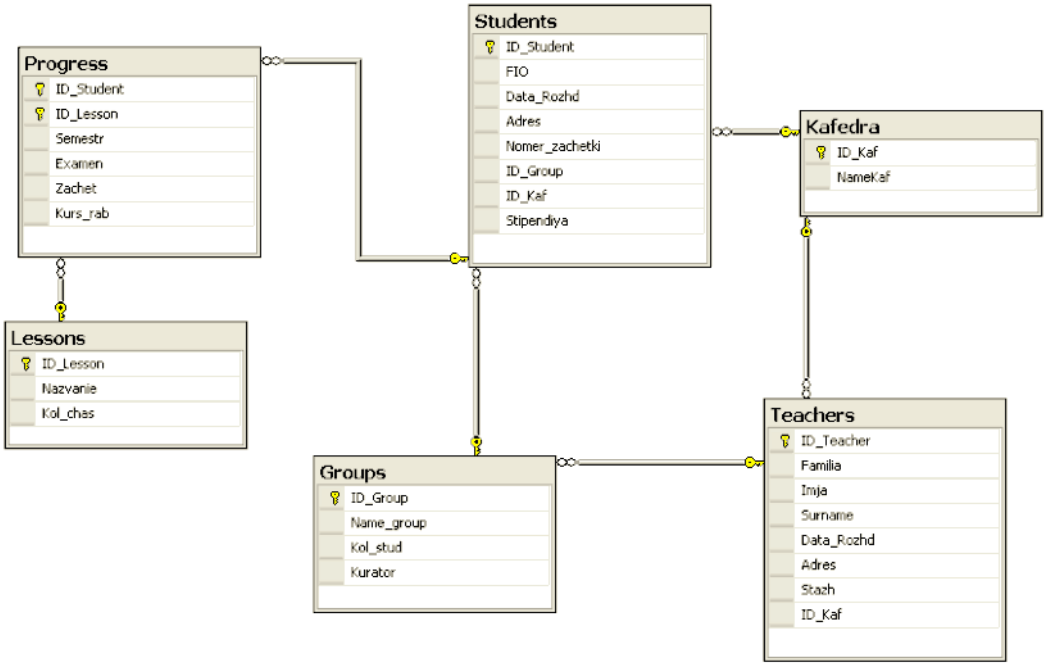


Рис. 5. Структура базы данных *Institute*

Перед созданием таблиц нужно указать базу данных, в которой будут создаваться требуемые таблицы, с помощью команды

USE имя_базы_данных;

В нашем случае это будет команда

USE Institute;

7.2. Ограничения целостности

Ограничение целостности – это набор определенных 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. В этом случае базовый синтаксис команды создания таблицы имеет следующий вид:

<определение_таблицы> ::=

CREATE TABLE имя_таблицы

(

{

имя_столбца тип_данных [NOT NULL] [UNIQUE]

[DEFAULT <значение>]

[CHECK (<условие_выбора>)] [,…n]

}

[CONSTRAINT имя_ограничения]

[PRIMARY KEY (имя_столбца [,…n])

{ [UNIQUE (имя_столбца [,…n]) }

[FOREIGN KEY (имя_столбца_внешнего_ключа [,…n])

REFERENCES имя_род_таблицы
[(имя_столбца_род_таблицы [,…n])],

[ON UPDATE {CASCADE | NO ACTION }]

[ON DELETE {CASCADE | NO ACTION }]

{ [CHECK (<условие_выбора>)] [,…n] }

);

Ограничения целостности имеют приоритет над триггерами, правилами и значениями по умолчанию.

7.2.1. Синтаксис ограничений целостности

Общий синтаксис ограничений целостности:

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

[предикат] [откладывание_ограничения] [время_ откладывания]

CONSTRAINT [имя_ограничения] – начинает определение ограничения и задает ограничению имя. Если имя не задано, то система создаст имя автоматически. Лучше задавать ограничениям осмысленные имена. В этом случае при выдаче системой сообщения о нарушении установленного ограничения будет указано его имя, а это упрощает обнаружение ошибок.

тип_ограничения – к ограничениям целостности относятся:

– ограничение первичного ключа ***PRIMARY KEY***;

– ограничение внешнего ключа ***FOREIGN KEY***;

– ограничение уникальности ***UNIQUE***;

– ограничение значения ***NULL***;

– ограничение на проверку ***CHECK***.

столбец [,…] связывает с ограничениями один или несколько столбцов. Столбцы перечисляются через запятую. Список столбцов следует заключать в скобки;

предикат определяет предикат для ограничений типа ***CHECK***;

откладывание_ограничения определяет для ограничения тип ***DEFERRABLE*** (допускающий откладывание) или ***NOT DEFERRABLE*** (не допускающий откладывание). Если ограничение может быть с отложенной проверкой, то можно указать, чтобы проверка нарушения правил производилась в конце транзакции. Если ограничение не допускает откладывния, то выполнение правил проверяется после каждой инструкции SQL;

время_откладывания – для ограничений с отложенной проверкой – определяется, является ли оно изначально откладываемым (***INITIALLY DEFERRED***) или изначально безотлагательным. Для изначально откладываемого ограничения время проверки сдвигается до конца транзакции, даже если она состоит из множества инструкций SQL. Для изначально безотлагательного ограничения проверка ограничения производится в конце каждой инструкции SQL.

7.2.2. Ограничение первичных ключей

Первичный ключ – атрибут или набор атрибутов, однозначно определяющих объект. Первичные ключи таблицы — это 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;

Пример 12. Создание таблицы ***Kafedra*** с ограничением первичного ключа.

```
CREATE TABLE Kafedra
(
ID_Kaf INTEGER PRIMARY KEY CHECK (ID_Kaf>=1 AND ID_Kaf<=6),
 NameKaf CHAR(7)   NOT NULL
);
```

Пример 13. Создание таблицы ***Lessons*** с ограничениями.

CREATE TABLE Lessons

(

ID_Lesson INT IDENTITY(1,1)

CONSTRAINT a_lesson PRIMARY KEY

CHECK (ID_Lesson BETWEEN 0 AND 999),

Nazvanie VARCHAR(50) NOT NULL

Kol_chas INT NOT NULL CHECK(Kol_chas BETWEEN 0 AND 999)

);

В этом примере ***a_lesson*** – это имя, присвоенное ограничению таблицы.

Пример 14. Создание таблицы Teachers

CREATE TABLE Teachers

(ID_Teacher INT IDENTITY(1,1) CONSTRAINT a_teacher PRIMARY KEY

CHECK (ID_Teacher BETWEEN 0 AND 9999),

Familia VARCHAR(20) NOT NULL,

Imja VARCHAR(20) NOT NULL,

Surname VARCHAR(20) NOT NULL,

Data_Rozhd DATETIME,

Adres VARCHAR(50),

Stazh TINYINT NOT NULL CHECK(Stazh BETWEEN 0 AND 99),

ID_Kaf INTEGER FOREIGN KEY CHECK (ID_Kaf>=1 AND ID_Kaf<=6),

);

В этом примере ***a_teacher*** **–** это имя, присвоенное ограничению таблицы.

7.2.3. Составные первичные ключи

Ограничение ***PRIMARY KEY*** может быть также применено для нескольких полей, составляющих уникальную комбинацию значений – составной первичный ключ. Рассмотрим таблицу ***Progress***. Очевидно, что ни к полю идентификатора студента (***ID_student***), ни

к полю идентификатора предмета обучения (***ID_Lesson***) по отдельности нельзя предъявить требование уникальности. Однако для того, чтобы в таблице не могли появиться разные записи для одинаковых комбинаций значений полей ***ID_student*** и ***ID_Lesson*** (конкретный студент на конкретном экзамене не может получить более одной оценки), имеет смысл объявить уникальной комбинацию этих полей. Для этого можно применить ограничение ***PRIMARY KEY***, объявив пару ***ID_student*** и ***ID_Lesson*** первичным ключом таблицы.

Пример 15. Создание таблицы ***Progress***:

```
CREATE TABLE Progress
(
    ID_Student INT NOT NULL CONSTRAINT to_student
        REFERENCES Students(ID_Student),
    ID_Lesson INT NOT NULL CONSTRAINT to_lesson
        REFERENCES Lessons(ID_Lesson),
    Semestr INT NOT NULL
        CHECK(Semestr BETWEEN 1 AND 10),
    Examen INT NOT NULL
        CHECK(Examen BETWEEN 2 AND 5),
    Zachet VARCHAR(10),
    Kurs_rab   TINYINT,
    CONSTRAINT a_progress PRIMARY KEY(ID_Student,
ID_Lesson));
```

Составные ключи указываются через запятую после последнего поля:

```
    primary key(ID_Student,ID_Lesson),
    );
```

7.2.4. Ограничение внешних ключей

Внешний ключ – набор атрибутов, содержащий ссылки на первичный ключ другого (или того же самого) отношения. Внешний ключ используется для поддержания ссылочной целостности, так как предотвращает ввод в таблицу данных, для которых нет соответствующих значений в связанной таблице.

Синтаксис:

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

Требования к внешнему ключу:

– 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.***

Пример 16. Создание таблицы ***Groups*** с ограничениями уровня столбца:

```
CREATE TABLE Groups
(
    ID_Group INT
        IDENTITY(1,1)
        CONSTRAINT a_group PRIMARY KEY
        CHECK (ID_Group BETWEEN 0 AND 999),
    Name_group VARCHAR(50) NOT NULL,
    Kol_stud INT NULL
        CHECK(Kol_stud BETWEEN 20 AND 30),
    Kurator INT NOT NULL
     CONSTRAINT to_kurator
     REFERENCES Teachers(ID_Teacher) );
```

Пример 17. Создание таблицы ***Groups*** с ограничениями уровня таблицы:

```
CREATE TABLE Groups
(
    ID_Group INT IDENTITY(1,1)
            CHECK (ID_Group BETWEEN 0 AND 999),
    Name_group VARCHAR(50) NOT NULL,
    Kol_stud INT NULL
        CHECK(Kol_stud BETWEEN 20 AND 30),
```

Kurator INT NOT NULL,
CONSTRAINT b_group PRIMARY KEY (ID_Group),
CONSTRAINT b_kurator FOREIGN KEY (Kurator)
REFERENCES Teachers(ID_Teacher)
);

7.2.5. Ограничение уникальности UNIQUE

UNIQUE – уникальное значение поля в пределах столбца таблицы. Если при создании таблицы для столбца указывается ограничение ***UNIQUE***, то база данных отвергает любую попытку ввести в это поле какой-либо строки значение, уже содержащееся в том же поле другой строки. Поля, созданные с ограничением ***UNIQUE,*** называют потенциальными ключами.

Это ограничение применимо только к тем полям, которые были объявлены ***NOT NULL.***

7.2.6. Ограничение значения NULL

NULL используется для указания того, что в данном столбце могут содержаться значения ***NULL,*** т.е. данные недоступны, опущены или недопустимы. Если указано ключевое слово ***NOT NULL***, то будут отклонены любые попытки поместить значение ***NULL*** в данный столбец.

7.2.7. Ограничение на проверку CHECK

CHECK (<условие>) используется для проверки допустимости данных, вводимых в конкретный столбец таблицы. Это еще один уровень защиты данных. ***CHECK*** задает диапазон возможных значений для столбца или столбцов.

Синтаксис ограничения:

CONSTRAINT [имя_ограничения] CHECK (условие_поиска)

условие_поиска – задаются ограничения на значения, вставляемые в столбец или таблицу.

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

менение одного и того же ограничения к разным столбцам и использование в логических выражениях значений других столбцов.

Считается, что значение удовлетворяет ограничению ***CHECK***, если результатом проверки является значение ***TRUE***.

7.2.8. Ограничение на значение по умолчанию

DEFAULT <выражение> – задание значения поля по умолчанию.

В Transact SQL есть возможность при вставке строки в таблицу, не указывая значений некоторого поля, определить значение этого поля по умолчанию. Значение может представлять собой константу (строку или число) или системную функцию, например, ***GETDATE().*** Наиболее часто используемым значением по умолчанию является ***NULL***. Это значение принимается по умолчанию для любого столбца, для которого не было установлено ограничение ***NOT NULL***.

Строго говоря, опция ***DEFAULT*** не имеет ограничительного свойства, так как она не ограничивает значения, вводимые в поле, а просто конкретизирует значение поля в случае, если оно не было задано.

Предположим, что у основной части групп, информация о которых находится в таблице ***Groups***, значение поля ***Kol_stud*** (количество студентов в группе) равно 25. Чтобы при задании атрибутов не вводить для большинства групп значение поля ***Kol_stud=25***, можно установить его как значение поля ***Kol_stud*** по умолчанию, определив таблицу ***Groups*** следующим образом:

```
CREATE TABLE Groups
(
    ID_Group INT IDENTITY(1,1)
            CHECK (ID_Group BETWEEN 0 AND 999),
    Name_group VARCHAR(50) NOT NULL,
    Kol_stud INT NULL DEFAULT 25
        CHECK(Kol_stud BETWEEN 15 AND 30),
    Kurator INT NOT NULL,
CONSTRAINT b_group PRIMARY KEY (ID_Group ),
CONSTRAINT b_kurator FOREIGN KEY (Kurator)
    REFERENCES Teachers(ID_Teacher)
);
```

Другая цель применения значения по умолчанию – это использование его как альтернативы для ***NULL***. ***NULL*** в качестве возможных значений поля существенно усложняет интерпретацию операций сравнения, в которых участвуют значения таких полей, поскольку ***NULL*** представляет собой признак того, что фактическое значение поля неизвестно или неопределенно. Следовательно, сравнение с ним любого конкретного значения (в рамках двузначной булевой логики) является некорректным. Исключение составляет специальная операция сравнения ***IS NULL***, которая определяет, является ли содержимое поля каким-либо значением или оно отсутствует.

Во многих случаях использование вместо ***NULL*** значения, подставляемого в поле по умолчанию, может существенно упростить использование значений поля в предикатах.

Например, можно установить для столбца опцию ***NOT NULL***, а для неопределенных значений числового типа установить значение по умолчанию «равно нулю», или для полей типа CHAR – пробел, использование которых в операциях сравнения не вызывает никаких проблем.

Пример 18. Создание таблицы ***Students***:

CREATE TABLE Students
(***ID_Student*** ***INT IDENTITY(1,1) PRIMARY KEY,***
Fio VARCHAR(70) NOT NULL,
Data_Rozhd DATETIME,
Adres ***VARCHAR(100,***
Nomer_zachetki VARCHAR(15) NOT NULL,
ID_Group INT NOT NULL CONSTRAINT to_group FOREIGN KEY REFERENCES Groups(ID_Group),
ID_Kaf INT FOREIGN KEY REFERENCES Kafedra(ID_Kaf)
);

7.2.9. Общие ограничения целостности

<***ограничения целостности таблицы***> – то же, что и для поля.

Общие ограничения целостности указываются через запятую после последнего поля.

7.3. Изменение таблицы

Структура существующей таблицы может быть модифицирована с помощью команды ***ALTER TABLE,*** упрощенный синтаксис которой представлен ниже:

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]

};

Команда позволяет добавлять и удалять столбцы, изменять их определения.

Одно из основных правил при добавлении столбцов в существующую таблицу гласит: если в таблице уже содержатся данные, добавляемый столбец не может быть определен с атрибутом ***NOT NULL***. Этот атрибут означает, что для каждой строки данных соответствующий столбец должен содержать некоторое значение, поэтому добавление столбца с атрибутом ***NOT NULL*** приводит к появлению противоречия – уже существующие строки данных таблицы не будут иметь в новом столбце ненулевых значений.

Способ добавления обязательных полей в существующую таблицу:

– добавить в таблицу новый столбец, определив его с атрибутом ***NULL*** (т.е. столбец не обязан содержать каких-либо значений);

– ввести в новый столбец какие-либо значения для каждой строки данных таблицы;

– убедившись, что новый столбец содержит ненулевые значения для каждой строки данных, изменить структуру таблицы, заменив атрибут этого столбца на ***NOT NULL***.

Правила изменения определений столбцов:

– размер столбца может быть увеличен до максимального значения, допускаемого соответствующим типом данных;

– размер столбца может быть уменьшен только в том случае, если содержащееся в нем наибольшее значение не будет превосходить его нового размера;

– количество разрядов числового типа данных всегда может быть увеличено;

– количество разрядов числового типа данных может быть уменьшено только в том случае, если количество разрядов наибольшего значения в соответствующем столбце не будет превосходить нового числа разрядов, определенного для этого столбца;

– количество десятичных знаков числового типа данных может быть уменьшено или увеличено;

– тип данных столбца, как правило, может быть изменен.

Пример 19. Добавить в таблицу ***Students*** поле ***Stipendiya***:

ALTER TABLE Students ADD Stipendiya INT;

Некоторые реализации фактически могут ограничить разработчика в использовании некоторых опций команды ***ALTER TABLE***. Например, может оказаться недопустимым удаление столбцов из существующей таблицы. Чтобы добиться этого, сначала потребуется удалить саму таблицу и только потом заново ее построить с нужными столбцами. Причем уже внесенные в таблицу данные будут потеряны.

Возможны трудности, связанные с удалением из таблицы столбца, который зависит от некоторого столбца другой таблицы. В таком случае сначала придется удалить ограничение столбца, а затем сам столбец.

Пример 20. Удаление ограничения внешнего ключа:

ALTER TABLE Students DROP CONSTRAINT to_group
ALTER TABLE Students DROP COLUMN ID_Group;

7.4. Удаление таблицы

С течением времени структура базы данных меняется: создаются новые таблицы, а прежние становятся ненужными и удаляются из базы данных с помощью команды

DROP TABLE имя_таблицы [RESTRICT | CASCADE;

Пример 21. Удалить таблицу Студенты:

DROP TABLE Students;

Команда ***DROP TABLE*** дополнительно позволяет указывать, следует ли операцию удаления выполнять каскадно.

Если указано ключевое слово ***RESTRICT***, то при наличии в базе данных хотя бы одного объекта, существование которого зависит от удаляемой таблицы, выполнение команды ***DROP TABLE*** будет отменено.

Если указано ключевое слово ***CASCADE***, автоматически удаляются и все прочие объекты базы данных, чье существование зависит от удаляемой таблицы, а также другие объекты, зависящие от удаляемых объектов. Общий эффект от выполнения команды ***DROP TABLE*** с ключевым словом ***CASCADE*** может оказаться весьма ощутимым, поэтому подобные операторы следует использовать с максимальной осторожностью.

Чаще всего команда ***DROP TABLE*** используется для исправления ошибок, допущенных при создании таблицы. Если таблица была создана с некорректной структурой, можно воспользоваться оператором ***DROP TABLE*** для ее удаления, после чего создать таблицу заново.

Следует отметить, что эта команда удалит и указанную таблицу, и все входящие в нее строки данных. Если требуется удалить из таблицы лишь данные, сохранив структуру таблицы, следует воспользоваться командой

TRUNCATE TABLE имя_таблицы

Эта команда делает то же самое, что и ***DELETE FROM***, но быстрее.

Пример 22. Удалить данные таблицы:

TRUNCATE TABLE Students;

Краткие итоги

Таблица – основной объект для хранения информации в реляционной базе данных.

Таблицы базы данных создаются с помощью команды ***CREATE TABLE***. Эта команда создает пустую таблицу, т.е. таблицу, не имеющую строк. Значения в эту таблицу вводятся с помощью команды ***INSERT***. Команда ***CREATE TABLE*** определяет имя таблицы и множество поименованных столбцов в указанном порядке. Для каждого столбца должен быть определен тип и размер.

Перед созданием таблиц нужно указать базу данных, в которой будут создаваться требуемые таблицы, с помощью команды

USE имя_базы_данных;

Ограничения целостности. Ограничение целостности – это набор определенных правил, которые устанавливают допустимость данных и связей между ними в любой момент времени.

К ограничениям целостности относятся:

– ограничение первичного ключа ***PRIMARY KEY***;

– ограничение внешнего ключа ***FOREIGN KEY***;

– ограничение уникальности ***UNIQUE***;

– ограничение значения ***NULL***;

– ограничение на проверку ***CHECK***.

Структура существующей таблицы может быть модифицирована с помощью команды ***ALTER TABLE.*** Команда позволяет добавлять и удалять столбцы, изменять их определения.

С течением времени структура базы данных меняется: создаются новые таблицы, а прежние становятся ненужными и удаляются из *БД* с помощью команды:

DROP TABLE имя_таблицы [RESTRICT | CASCADE]

Следует отметить, что эта команда удалит и указанную таблицу, и все входящие в нее строки данных. Если требуется удалить из таблицы лишь данные, сохранив структуру таблицы, следует воспользоваться командой

TRUNCATE TABLE имя_таблицы;

Контрольные вопросы

1. С помощью какой команды можно создать таблицу базы данных?
2. Для какого типа данных при создании таблицы обязательно должен быть указан размер?
3. Что означает свойство ***IDENTITY*** ?
4. Какие ограничения целостности могут быть заданы при создании таблицы?
5. Что такое первичный ключ?
6. Что такое внешний ключ?
7. Каковы особенности первичных и внешних ключей?
8. Что означает ограничение ***NULL***?
9. Для чего используется ограничение ***CHECK***?

8. Команды модификации данных

К командам модификации данных относятся:

– ***INSERT INTO*** – команда добавления;

– ***DELETE FROM*** – команда удаления;

– ***UPDATE*** – команда обновления.

8.1. Команда добавления

Оператор ***INSERT INTO*** применяется для добавления записей в таблицу. Формат команды:

< команда добавления >::=

INSERT INTO <имя_таблицы> [(имя_столбца [,...n])]

VALUES (значение[,...n]);

где ***имя_таблицы*** представляет собой либо имя таблицы базы данных, либо имя обновляемого представления.

Эта форма команды ***INSERT*** с параметром ***VALUES*** предназначена для вставки единственной строки в указанную таблицу.

Список столбцов указывает столбцы, которым будут присвоены значения в добавляемых записях.

Список может быть опущен. Тогда подразумеваются все столбцы таблицы (кроме объявленных как счетчик), причем в порядке, установленном при создании таблицы.

Если в команде ***INSERT*** указывается конкретный список имен полей, то любые пропущенные в нем столбцы должны быть объявлены при создании таблицы как допускающие значение ***NULL***, за исключением тех случаев, когда при описании столбца использовался параметр ***DEFAULT***.

Список значений должен соответствовать списку столбцов следующим образом:

1) количество элементов в обоих списках должно быть одинаковым;

2) должно существовать прямое соответствие между позицией одного и то же элемента в обоих списках. Поэтому I элемент списка значений должен относиться к I столбцу в списке столбцов, II – ко II столбцу и т.д.

3) типы данных элементов в списке значений должны быть совместимы с типами данных соответствующих столбцов таблицы.

Пример 23. Добавить в таблицу ***Teachers*** новую запись:

INSERT INTO Teachers (Familia, Imja, Surname, Data_Rozhd, Adres, Stazh, ID_Kaf)

VALUES ('Гурьянов', 'Лев', 'Вячеславович ','1952-07-07,' ул. Комсомольская, д.24 кв. 26, 30,1);

Если столбцы таблицы ***Teachers*** указаны в полном составе и в том порядке, в котором они перечислены при создании таблицы ***Teachers***, команду можно упростить.

Пример 24. Добавить в таблицу ***Teachers*** новую запись:

INSERT INTO Teachers VALUES ('Гурьянов', 'Лев', 'Вячеславович ','1952-07-07,' ул. Комсомольская, д.24 кв. 26, 30,1);

После добавления записей таблицы будут иметь приведенный на рис. 6.

Таблица ***Kafedra***

ID_Kaf	NameKaf
1	МОиПЭВМ
2	САПР
3	ИнОУп
4	ВТ

Таблица ***Groups***

ID_Group	Name_group	Kol_stud	Kurator
11	06ВП2	20	7
12	06ВП1	25	8
13	06ВВ1	20	9
14	07ВП1	23	10
15	07ВП2	24	11
16	08ВП1	23	12

Таблица ***Teachers***

ID_Teacher	Familia	Imja	Surname	Data_Rozhd	Adres	Stazh
7	Гурьянов	Лев	Вячеславович	23.04.1905 0:0...	ул. Комсомольс...	30
8	Шашков	Борис	Дмитриевич	18.04.1905 0:0...	ул. Ладожская,...	35
9	Сивохин	Александр	Васильевич	23.04.1905 0:0...	ул. Онежская, ...	30
10	Шибанов	Сергей	Владимирович	09.05.1905 0:0...	ул. Плеханова,...	28
11	Самуйлов	Сергей	Владимирович	11.05.1905 0:0...	ул. Кижеватов...	30
12	Мещеряков	Борис	Кузьмич	23.04.1905 0:0...	ул. Кижеватов...	40

Таблица ***Students***

ID_Student	FIO	Data_Rozhd	Adres	Nomer_zachetki	ID_Group	ID_Kaf
74	Иванков С.В.	23.12.1990 0:0...	NULL	08ВП129	12	1
75	Буртасов И.Ю.	12.03.1990 0:0...	NULL	08ВП102	12	1
76	Трапин А.А.	24.09.1990 0:0...	NULL	08ВП201	13	1
77	Глинин И.В.	17.07.1989 0:0...	NULL	08ВП203	13	1
78	Панин С.С.	25.06.1990 0:0...	NULL	08ВП204	14	1
79	Илюхин В.И.	29.03.1988 0:0...	NULL	08ВП103	12	1
80	Макарь В.А.	23.01.1991 0:0...	NULL	06ВП118	14	1
81	Карпов А.В.	17.02.1989 0:0...	NULL	06ВП111	14	1

Рис. 6. Заполнение таблицы базы данных (начало)

Таблица ***Lessons***

ID_Lesson	Nazvanie	Kol_chas
1	Объектно-орие...	68
2	Компьютерная ...	68
3	Операционные ...	51
4	Организация ЭВМ	51
5	Алгебра и геом...	34
6	Алгоритмическ...	68
7	Физика	68
8	Начертательна...	68
9	История техники	68
10	Иностранный я...	51

Таблица ***Progress***

ID_Student	ID_Lesson	Semestr	Examen	Zachet	Kur
74	1	4	5	NULL	NU
75	1	4	3	NULL	NU
79	1	4	5	NULL	NU
79	3	6	5	NULL	NU
79	4	5	5	NULL	NU
81	1	4	5	NULL	NU
81	3	6	5	NULL	NU
81	4	5	4	NULL	NU
83	3	6	5	NULL	NU
83	4	5	4	NULL	NU
83	20	4	NULL	зачтено	NU
83	23	6	5	NULL	NU
83	24	6	5	зачтено	5
83	25	6	4	NULL	NU
84	1	4	5	NULL	NU
84	3	6	5	NULL	NU

Рис. 6. Окончание

8.2. Команда удаления

Команда ***DELETE FROM*** удаляет данные из таблицы:

DELETE FROM <имя_таблицы>

[WHERE <условие_отбора>];

имя_таблицы представляет собой либо имя таблицы базы данных, либо имя обновляемого представления.

условие_отбора – здесь задается условие, в соответствии с которым будут удаляться записи (подробнее см. подразд. 9.2).

Если предложение ***WHERE*** присутствует, удаляются записи из таблицы, удовлетворяющие условию отбора. Если опустить предложение ***WHERE***, из таблицы будут удалены все записи без предупреждения и без запроса на подтверждения, однако сама таблица сохранится.

Пример 25. Удалить все предметы, на которые отводится 85 часов:

DELETE

FROM Lessons

WHERE Kol_chas=68;

После выполнения этого оператора таблица ***Lessons*** будет иметь вид, представленный на рис. 7.

ID_Lesson	Nazvanie	Kol_chas
3	Операционные ...	51
4	Организация ЭВМ	51
5	Алгебра и геом...	34
10	Иностранный я...	51
11	Информатика	51
12	Отечественная...	51
13	Дискретная ма...	34
14	C++	34

Рис. 7. Вид таблицы *Lessons* после удаления строк

При удалении строк с помощью *DELETE* эти строки сохраняются в системных сегментах отката на случай восстановления. Это может потребовать значительного времени. Поэтому лучше использовать *TRUNCATE* для удаления всех данных.

8.3. Команда обновления

<оператор_изменения> ::=

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

[WHERE <условие_отбора>]

имя_таблицы – это либо имя таблицы базы данных, либо имя обновляемого представления.

В предложении ***SET*** указываются имена одного и более столбцов, данные в которых необходимо изменить.

Выражение представляет собой новое значение соответствующего столбца и должно быть совместимо с ним по типу данных.

Оператор ***UPDATE*** без предложения ***WHERE*** следует использовать с осторожностью, так как в этом случае будут затронуты все строки таблицы.

Пример 26. Увеличить стипендию, равную 1200 рублям, на 25 %:

UPDATE Students SET Stipend = Stipend*1.25

WHERE Stipend =1200;

После выполнения этого оператора таблица ***Students*** будет иметь вид, представленный на рис. 8.

ID_Student	FIO	Data_Rozhd	Adres	Nomer_zachetki	ID_Group	ID_Kaf	Stipendiya
74	Иванков С.В.	23.12.1990 0:0...	NULL	08ВП129	12	1	1200
75	Буртасов И.Ю.	12.03.1990 0:0...	NULL	08ВП102	12	1	1500
76	Трапин А.А.	24.09.1990 0:0...	NULL	08ВП201	13	1	NULL
77	Глинин И.В.	17.07.1989 0:0...	NULL	08ВП203	13	1	1500
78	Панин С.С.	25.06.1990 0:0...	NULL	08ВП204	14	1	NULL
79	Илюхин В.И.	29.03.1988 0:0...	NULL	08ВП103	12	1	NULL
80	Макарь В.А.	23.01.1991 0:0...	NULL	06ВП118	14	1	1500
81	Карпов А.В.	17.02.1989 0:0...	NULL	06ВП111	14	1	1700

Рис. 8. Вид таблицы *Students* после выполнения оператора *UPDATE*

Пример 27. Для студента Иванкова С.В. установить стипендию в размере 2000 рублей:

UPDATE Students SET Stipend=2000 WHERE FIO LIKE ' Иванкова С.В.;'

После выполнения этого оператора таблица ***Students*** будет иметь вид, представленный на рис. 9.

ID_Student	FIO	Data_Rozhd	Adres	Nomer_zachetki	ID_Group	ID_Kaf	Stipendiya
74	Иванков С.В.	23.12.1990 0:0...	NULL	08ВП129	12	1	2000
75	Буртасов И.Ю.	12.03.1990 0:0...	NULL	08ВП102	12	1	1500
76	Трапин А.А.	24.09.1990 0:0...	NULL	08ВП201	13	1	NULL
77	Глинин И.В.	17.07.1989 0:0...	NULL	08ВП203	13	1	1500
78	Панин С.С.	25.06.1990 0:0...	NULL	08ВП204	14	1	NULL
79	Илюхин В.И.	29.03.1988 0:0...	NULL	08ВП103	12	1	NULL
80	Макарь В.А.	23.01.1991 0:0...	NULL	06ВП118	14	1	1500
81	Карпов А.В.	17.02.1989 0:0...	NULL	06ВП111	14	1	1700

Рис. 9. Результат выполнения запроса

Пример 28. Увеличить максимальную стипендию в 2 раза:

UPDATE Students SET Stipend = Stipend*2
WHERE Stipend = (SELECT MAX(Stipend) FROM Students);

Краткие итоги

Существуют следующие операторы модификации данных:

- ***INSERT INTO*** – оператор добавления; применяется для добавления записей в таблицу:

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

- ***DELETE FROM*** – оператор удаления; применяется для удаления записей из таблицы.

DELETE FROM <имя_таблицы> [WHERE <условие_отбора>]

При удалении строк с помощью ***DELETE*** эти строки сохраняются в системных сегментах отката на случай восстановления. Это может потребовать значительного времени. Поэтому лучше использовать ***TRUNCATE*** для удаления всех данных.

- ***UPDATE*** – оператор обновления,

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

[WHERE <условие_отбора>]

Контрольные вопросы

1. Какие операторы ***TRANSACT SQL*** используются:

а) для вставки строки в таблицу;

б) удаления строки из таблицы;

в) обновления строки в таблице?

2. Какое соответствие должно быть между списком столбцов и списком значений в операторе ***INSERT***?

9. Команда SELECT

Команда ***SELECT*** – средство, которое полностью абстрагировано от вопросов представления данных, что помогает сконцентрировать внимание на проблемах доступа к данным.

Примеры использования ***SELECT*** наглядно демонстрируют один из основополагающих принципов больших (промышленных) СУБД: ***средства хранения данных и доступа к ним отделены от средств представления данных***.

Операции над данными производятся в ***масштабе наборов данных, а не отдельных записей.***

Один и тот же запрос может быть реализован несколькими способами, которые будучи правильными, тем не менее, могут существенно отличаться по времени исполнения, и это особенно важно для больших БД.

Команда SELECT не изменяет данные в базе данных, а только производит их выборку в соответствии с заданными критериями.

Команда ***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>]
[ORDER BY имя_столбца [,...n]];

Команда ***SELECT*** определяет поля (столбцы), которые будут входить в результат выполнения запроса. В списке они разделяются запятыми и приводятся в такой последовательности, в какой должны быть представлены в результате запроса. Если используется имя поля, содержащее пробелы или разделители, его следует заключить в квадратные скобки.

Если обрабатывается несколько таблиц, то (при наличии одноименных полей в разных таблицах) в списке полей используется полная спецификация поля, т.е. ***Имя_таблицы.Имя_поля.***

Порядок выполнения оператора ***SELECT***:

1. ***FROM*** – вначале определяются имена используемых таблиц;
2. ***WHERE*** – из указанной таблицы выбираются записи, удовлетворяющие заданным условиям;

3. ***GROUP BY*** – выполняется группировка полученных записей, т.е. образуются группы строк, имеющих одно и то же значение в указанном столбце;

4. ***HAVING*** – выбор группы строк, удовлетворяющих указанным условиям;

5. ***ORDER BY*** – выполняется сортировка записей в указанном порядке

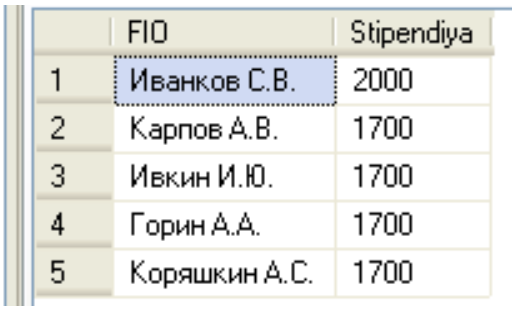
6. ***SELECT*** – устанавливается, какие столбцы должны присутствовать в выходных данных.

Порядок предложений и фраз в команде ***SELECT*** не может быть изменен. Только два предложения ***SELECT*** и ***FROM*** являются обязательными, все остальные могут быть опущены. ***SELECT*** – закрытая операция: ***результат запроса*** к таблице представляет собой другую таблицу (табл. 17).

Таблица 17

Описание предикатов оператора *SELECT*

Элемент	Описание
1	2
*****	Символом ***** можно выбрать все поля, а вместо имени поля применить выражение из нескольких имен
ALL	Если ***SELECT*** не содержит ни одного предиката, то подразумевается предикат ***ALL***. Отбираются все записи, соответствующие условиям, заданным в инструкции ***SELECT***. Приведенные ниже команды Transact SQL эквивалентны; они возвращают все записи из таблицы ***Students***: ***SELECT ALL FROM Students*** ***SELECT * FROM Students***
DISTINCT	Исключает записи, которые содержат повторяющиеся значения в выбранных полях. Чтобы запись была включена в результат выполнения запроса, значения в каждом поле, включенном в команду ***SELECT***, должны быть уникальными. Например, в таблице ***Students*** есть однофамильцы. Если две записи содержат значение Иванов в поле ***FIO***, то следующая команда возвратит только одну из них: ***SELECT DISTINCT FIO FROM Students***; Если опустить предикат ***DISTINCT***, этот запрос возвратит обе записи для фамилии Иванов. Результат выполнения команды ***SELECT***, содержащей предикат ***DISTINCT***, является необновляемым и не отражает последующие изменения, внесенные другими пользователями

1	2
TOP n [PERCENT]	Возвращает определенное число записей, находящихся в начале или в конце диапазона, описанного с помощью предложения ***ORDER BY***. Следующая команда позволяет получить список 5 студентов с самой высокой стипендией ***SELECT TOP 5 FIO, Stipendiya*** ***FROM Students ORDER BY Stipendiya DESC;*** Результат выполнения запроса приведен на рисунке.  Если предложение ***ORDER BY*** будет опущено, запрос возвратит произвольный набор 5 записей из таблицы ***Students***, удовлетворяющих предложению ***WHERE***. Можно использовать зарезервированное слово ***PERCENT*** для возврата определенного процента записей, находящихся в начале или в конце диапазона, описанного с помощью предложения ***ORDER BY***. Например, вместо 5 студентов с самой высокой стипендией следует отобрать студентов, попавших в последние 5 процентов: ***SELECT TOP 5 PERCENT FIO, Stipendiya*** ***FROM Students ORDER BY Stipendiya ASC;*** Предикат ***ASC*** обеспечивает возврат последних значений. Значение, следующее после предиката ***TOP***, должно быть ***числовым значением типа Integer без знака.*** Предикат ***TOP*** не влияет на возможность обновления запроса

	FIO	Stipendiya
1	Иванков С.В.	2000
2	Карпов А.В.	1700
3	Ивкин И.Ю.	1700
4	Горин А.А.	1700
5	Коряшкин А.С.	1700

9.1. Предложение FROM

FROM задает имена таблиц и просмотров, которые содержат поля, перечисленные в команде ***SELECT***. Необязательный параметр псевдонима – это сокращение, устанавливаемое для имени таблицы.

Пример 29. Составить список сведений обо всех студентах:

SELECT * FROM Students;

Пример 30. Составить список всех преподавателей:

SELECT ALL Familia, Imja , Surname FROM Teachers;

Или (что эквивалентно)

SELECT Familia, Imja , Surname FROM Teachers;

Результат выполнения запроса приведен на рис. 10.

1	Гурьянов	Лев	Вячеславович
2	Шашков	Борис	Дмитриевич
3	Сивохин	Александр	Васильевич
4	Шибанов	Сергей	Владимирович
5	Самуйлов	Сергей	Владимирович
6	Мещеряков	Борис	Кузьмич

Рис. 10. Список всех преподавателей

9.2. Предложение WHERE

Это предложение определяет, какие записи из таблиц, перечисленных в предложении ***FROM***, следует включить в результат выполнения команды ***SELECT***.

За ключевым словом ***WHERE*** следует перечень условий поиска, определяющих те строки, которые должны быть выбраны при выполнении запроса.

Предложение ***WHERE*** может содержать до 40 выражений, связанных логическими операторами, такими как ***AND*** и ***OR***.

Основные типы условий отбора (или предикатов):

1. ***Сравнение.***
2. ***Диапазон.***
3. ***Принадлежность множеству.***

4. ***Соответствие шаблону.***

5. ***Значение NULL.***

9.2.1. Сравнение

В этой операции сравниваются результаты вычисления одного выражения с результатами вычисления другого.

В языке SQL можно использовать следующие операторы сравнения:

= – равенство;

< – меньше;

> – больше;

<= – меньше или равно;

>= – больше или равно;

<> – не равно.

Более сложные предикаты могут быть построены с помощью логических операторов ***AND***, ***OR*** или ***NOT***, а также скобок, используемых для определения порядка вычисления выражения.

Правила вычисления выражения в условиях:

– выражение вычисляется слева направо;

– первыми вычисляются подвыражения в скобках;

– операторы ***NOT*** выполняются до выполнения операторов ***AND*** и ***OR***;

– операторы ***AND*** выполняются до выполнения операторов ***OR***.

Для устранения любой возможной неоднозначности рекомендуется использовать скобки.

Пример 31. Вывести список студентов, которые получают стипендию больше или равную 1500 и меньше или равную 2000:

SELECT FIO, Stipendiya

FROM Students

WHERE (Stipendiya>=1500) And (Stipendiya<=2000)

Результат выполнения запроса приведен на рис. 11.

	FIO	Stipendiya
1	Иванков С.В.	2000
2	Карпов А.В.	1700
3	Ивкин И.Ю.	1700
4	Горин А.А.	1700
5	Коряшкин А.С.	1700

Рис. 11. *SELECT* с условием

9.2.2. Диапазон

В этой операции проверяется, попадает ли результат вычисления выражения в заданный диапазон значений. Диапазон задается с помощью ключевого слова ***BETWEEN***.

Диапазон определяется своими минимальным и максимальным значениями. При этом указанные значения включаются в условие поиска.

Пример 32. Вывести список студентов, которые получают стипендию больше 1500 и меньше или равную 2000 (запрос эквивалентен примеру 31):

SELECT Fio, Stipendiya

FROM Students

WHERE Stipendiya BETWEEN 1500 AND 2000;

При использовании отрицания ***NOT BETWEEN*** требуется, чтобы проверяемое значение лежало вне границ заданного диапазона.

Пример 33. Вывести список студентов, у которых стипендия не попадает в диапазон от 1500 до 2000:

SELECT Fio, Stipendiya

FROM Students

WHERE Stipendiya NOT BETWEEN 1500 AND 2000;

9.2.3. Принадлежность множеству

Здесь проверяется, принадлежит ли результат вычислений выражения заданному множеству значений.

Оператор ***IN*** используется для сравнения некоторого значения со списком заданных значений. При этом проверяется, соответству-

ет ли результат вычисления выражения одному из значений в предоставленном списке. С помощью оператора ***IN*** может быть достигнут тот же результат, что и в случае применения оператора ***OR***, однако оператор ***IN*** выполняется быстрее.

NOT IN используется для отбора любых значений, кроме тех, которые указаны в представленном списке.

Пример 34. Вывести список преподавателей, у которых стаж работы составляет 30 или 35 лет:

SELECT Familia, Stazh
FROM Teachers
WHERE Stazh IN (30, 35);

Результат выполнения запроса приведен на рис. 12.

	Familia	Stazh
1	Гурьянов	30
2	Шашков	35
3	Сивохин	30
4	Самуйлов	30

Рис. 12. *SELECT* с оператором IN

NOT IN используется для отбора любых значений, кроме тех, которые указаны в представленном списке.

Пример 35. Вывести список преподавателей, у которых стаж работы не 30 или не 35 лет:

SELECT Familia, Stazh
FROM Teachers
WHERE Stazh NOT IN (30, 35);

Результат выполнения запроса приведен на рис. 13.

	Familia	Stazh
1	Шибанов	28
2	Мешеряков	40

Рис. 13. *SELECT* с оператором NOT IN

9.2.4. Соответствие шаблону

Здесь проверяется, отвечает ли некоторое строковое значение заданному шаблону.

С помощью оператора ***LIKE*** можно выполнять сравнение выражения с заданным шаблоном.

Символы-заменители, используемые в шаблоне:

– ***%*** – 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 36. Найти студентов, у которых в номере зачетной книжки четвертый символ ‘П’:

SELECT Fio, Nomer_zachetki

FROM Students

WHERE Nomer_zachetki LIKE ’___П%’;

Пример 37. Найти студентов, у которых в номере зачетной книжки второй символ ‘6’ или ‘8’:

SELECT Fio, Nomer_zachetki

FROM Students

WHERE Nomer_zachetki LIKE ’_[68]%’;

Пример 38. Найти студентов, у которых в номере зачетной книжки второй символ “6” ,‘7’ или ‘8’:

SELECT Fio, Nomer_zachetki

FROM Students

WHERE Nomer_zachetki LIKE ’_[678]%’;

Пример 39. Найти студентов, у которых в фамилии встречается слог ‘ко’***SELECT FIO:***

FROM Students

WHERE FIO LIKE ’%ко%’;

Результат выполнения запроса приведен на рис. 14.

	FIO
1	Иванков С.В.
2	Павликов А.А.
3	Иванчуков А.Г.
4	Коряшкин А.С.
5	Бирюков В.В.
6	Иваненков С.В.

Рис. 14. *SELECT* с проверкой соответствия шаблону

9.2.5. Значение NULL

Значение ***NULL*** проверяет, содержит ли данный столбец определитель *NULL* (неопределенное значение).

Оператор ***IS NULL*** используется для сравнения текущего значения со значением ***NULL*** – специальным значением, указывающим на отсутствие любого значения. ***NULL*** – это не то же самое, что знак пробела (пробел – допустимый символ) или нуль (0 – допустимое число). ***NULL*** отличается и от строки нулевой длины (пустой строки). ***IS NOT NULL*** используется для проверки присутствия значения в поле.

Пример 40. Найти студентов, которые не получают стипендию:

SELECT Fio, Stipendiya

FROM Students

WHERE Stipendiya IS NULL;

Пример 41. Найти студентов, которые получают стипендию:
SELECT Fio, Stipendiya

FROM Students

WHERE Stipendiya IS NOT NULL;

9.3. Предложение ORDER BY

ORDER BY сортирует данные 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 задается ключевым словом ***ASC***. Для выполнения сортировки в обратной последовательности необходимо указать

ключевое слово ***DESC***. Фраза ***ORDER BY*** позволяет упорядочить выбранные записи в порядке возрастания или убывания значений любого столбца или комбинации столбцов, независимо от того, присутствуют эти столбцы в таблице результата или нет. ***Фраза ORDER BY всегда должна быть последним элементом в операторе SELECT.***

Пример 42. Вывести список студентов в алфавитном порядке:

SELECT Fio

FROM Students

ORDER BY Fio;

Во фразе ***ORDER BY*** может быть указано и больше одного элемента. Главный (первый) ключ сортировки определяет общую упорядоченность строк результирующей таблицы. Если во всех строках результирующей таблицы значения главного ключа сортировки являются уникальными, нет необходимости использовать дополнительные ключи сортировки. Однако если значения главного ключа не уникальны, в результирующей таблице будет присутствовать несколько строк с одним и тем же значением старшего ключа сортировки. В этом случае, возможно, придется упорядочить строки с одним и тем же значением главного ключа по какому-либо дополнительному ключу сортировки.

Пример 43. Вывести ФИО студентов и номера их зачеток. Фамилии студентов упорядочить в алфавитном порядке, а номера зачеток – по убыванию:

SELECT Fio, Nomer_zachetki

FROM Students

ORDER BY Fio, Nomer_zachetki DESC;

Пример 44. Упорядочить вывод по номеру столбца:

SELECT Fio, Data_Rozhd, Nomer_zachetki, Adres FROM students ORDER BY 3 DESC;

Результат выполнения запроса приведен на рис. 15.

	FIO	Data_Rozhd	Nomer_zachetki
1	Панин С.С.	1990-06-25 00:00:00.000	08ВП204
2	Глинин И.В.	1989-07-17 00:00:00.000	08ВП203
3	Трапин А.А.	1990-09-24 00:00:00.000	08ВП201
4	Иванков С.В.	1990-12-23 00:00:00.000	08ВП129

Рис. 15. Упорядочение вывода

Пример 45. Использование строковых констант в списке вывода**.** Получить список студентов с пояснительным текстом

SELECT Nomer_zachetki + ’ ’ + Студент ’ + Fio FROM Students

Результат выполнения запроса приведен на рис. 16.

	(No column name)	
1	08ВП129	Студент Иванков С.В.
2	08ВП102	Студент Буртасов И.Ю.
3	08ВП201	Студент Трапин А.А.
4	08ВП203	Студент Глинин И.В.
5	08ВП204	Студент Панин С.С.
6	08ВП103	Студент Илюхин В.И.
7	06ВП118	Студент Макарь В.А.
8	06ВП111	Студент Каропов А.В.

Рис. 16. Использование строковых констант в списке вывода

9.4. Использование итоговых функций

С помощью итоговых (агрегатных) функций в SQL-запросе можно получить ряд обобщающих статистических сведений о множестве отобранных значений выходного набора.

Пользователю доступны следующие основные итоговые 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 определенном поле записей, отобранных запросом. Оно является арифметическим средним значением, т.е. суммой значений, деленной на их количество.

– ***Sum (Выражение)*** – вычисляет сумму множества значений, содержащихся в определенном поле записей, отобранных запросом.

Чаще всего в качестве ***выражения*** выступают ***имена столбцов***. Выражение может вычисляться и по значениям нескольких таблиц.

Все эти 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.

Вариант ***COUNT(*)*** – особый случай использования функции ***COUNT.*** Назначение – подсчет всех строк в результирующей таблице, независимо от того, содержатся там пустые, дублирующиеся или любые другие значения.

Если до применения обобщающей функции необходимо исключить дублирующиеся значения, следует перед именем столбца в определении функции поместить ключевое слово ***DISTINCT***. Оно не имеет смысла для функций ***MIN*** и ***MAX***, однако его использование может повлиять на результаты выполнения функций ***SUM*** и ***AVG.*** Необходимо заранее обдумать, должно ли оно присутствовать в каждом конкретном случае. Ключевое слово ***DISTINCT*** может быть указано в любом запросе не более 1 раза.

Особые случаи применения итоговых функций:

SUM (DISTINCT <поле>) – суммирование различных значений поля;

AVG (DISTINCT <поле>) – среднее арифметическое разных значений поля;

COUNT (DISTINCT <поле>) – подсчет количества разных значений поля;

COUNT (<поле>) – подсчет количества ненулевых значений поля;

COUNT (*) – подсчет количества строк в результате.

Итоговые функции могут использоваться только в списке предложения ***SELECT*** и в составе предложения ***HAVING***. Во всех других случаях это недопустимо.

Пример 46. Определить максимальное количество студентов в группах:

SELECT MAX (Kol_stud)

FROM Groups;

Пример 47. Определить количество различных групп (без повторений) в отношении ***Students*:**

SELECT COUNT (DISTINCT ID_Group) AS [Количество групп] FROM Students;

Результат выполнения запроса приведен на рис. 17.

	Количество групп
1	4

Рис. 17. *SELECT* без повторений

Пример 48. Определить количество групп (с повторениями) в отношении Студент ***SELECT COUNT (ID_Group) AS [Количество групп] FROM Students***

Результат выполнения запроса приведен на рис. 18.

Количество групп
23

Рис. 18. *SELECT* с повторениями

Пример 49. Определить первую по алфавиту фамилию студента

SELECT Min(FIO) AS Min_Фамилия FROM Students

Результат выполнения запроса приведен на рис. 19.

	Min_Фамилия
1	Абышкин В.В.

Рис. 19. Применение функции *Min* к текстовому полю

Пример 50. Определить общее количество студентов

SELECT COUNT (*) FROM Students

Пример 51. Определить количество студентов гр. 08ВП2

SELECT Kol_stud FROM Groups

WHERE Name_group LIKE '08ВП2'

Пример 52. Подсчитать среднюю стипендию студентов

SELECT AVG (Stipendiya) FROM Students

Пример 53. Определить суммарную стипендию студентов

SELECT SUM (Stipendiya) FROM Students

Результат выполнения запроса приведен на рис. 20

	Суммарная стипендия
1	23550

Рис. 20. Применение функции *SUM*

9.5. Предложение GROUP BY

Группирование данных – это размещение данных в столбцах с повторяющимися значениями в определенном логическом порядке.

Например, в базе данных содержится информация о студентах. Студенты учатся в разных группах. Вполне вероятно, что может понадобиться информация по каждой группе и обучающихся там студентах. Для этого следует сгруппировать информацию о студентах по группам.

Предположим, что необходимо найти среднюю стипендию студентов по каждой группе. Это можно сделать, применив к столбцу ***Stipendiya*** итоговую функцию ***AVG***, а затем использовать ***GROUP BY*** для группирования выводимых данных по группам.

Запрос, в котором присутствует ***GROUP BY***, называется ***группирующим запросом***. В нем группируются данные, полученные в результате выполнения команды ***SELECT***, после чего для каждой отдельной группы создается единственная суммарная строка.

Стандарт SQL требует, чтобы команда ***SELECT*** и фраза ***GROUP BY*** были тесно связаны между собой. При наличии в команде ***SELECT*** фразы ***GROUP BY*** каждый элемент списка дол-

жен иметь единственное значение для всей группы. Более того, команда ***SELECT*** может включать только следующие типы элементов:

– имена полей;

– итоговые функции;

– константы;

– выражения, включающие комбинации перечисленных выше элементов.

Все имена полей, приведенные в списке предложения ***SELECT***, должны присутствовать и во фразе ***GROUP BY*** – за исключением случаев, когда имя столбца используется в итоговой функции. Однако во фразе ***GROUP BY*** могут быть имена столбцов, отсутствующие в списке предложения ***SELECT*** (***но данные из этих столбцов не выводятся!***)

Если совместно с ***GROUP BY*** используется предложение ***WHERE***, то оно обрабатывается первым, а группированию подвергаются только те строки, которые удовлетворяют условию поиска.

Стандартом SQL определено, что при проведении группирования все отсутствующие значения рассматриваются как равные. Если две строки таблицы в одном и том же группируемом столбце содержат значение ***NULL*** и идентичные значения во всех остальных непустых группируемых столбцах, они помещаются в одну и ту же группу.

Пример 54. Определить максимальную и минимальную стипендии у студентов каждой группы:

SELECT ID_Group, MAX(Stipendiya) AS Максимальная_стипендия, MIN(Stipendiya) AS Максимальная_стипендия

FROM Students

GROUP BY ID_Group;

Результат выполнения запроса приведен на рис. 21.

	Номер_группы	Максимальная_стипенд...	Минимальная_стипендия
1	11	1700	1500
2	12	2000	1250
3	13	1500	1500
4	14	1700	1500

Рис. 21. Применение группировки в операторе *SELECT*

Пример 55. Вычислить среднюю стипендию по каждой группе:

SELECT Groups.Name_group, AVG(Students.Stipend)
AS ' Средняя стиипендия'
FROM Groups, Students
WHERE Students.ID_Group=Groups.ID_Group
GROUP BY Groups.Name_group;

Результат выполнения запроса приведен на рис. 22.

	Name_group	Средняя стипендия
1	06ВВ1	1500
2	06ВП1	1583
3	06ВП2	1580
4	07ВП1	1566

Рис. 22. Применение группировки в операторе *SELECT*

9.6. Предложение HAVING

Это дополнительная возможность «профильтровать» выходной набор.

Предложение ***HAVING*** работает следующим образом:

1) ***GROUP BY*** разделяет строки на наборы (по типу);

2) на полученные группы накладываются условия предложения ***HAVING***.

Сравнение условий в ***HAVING*** и условий в ***WHERE***:

– ***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.

Пример 56. Вывести список групп, в которых общее количество экзаменов больше 3:

SELECT gr.Name_group AS Группа, COUNT (DISTINCT (pr.ID_Lesson)) AS Количество экзаменов

FROM Groups gr, Progress pr, Students st

WHERE st.ID_Student= pr.ID_Student AND gr.ID_Group=st.ID_Group

AND EXISTS

(SELECT Examen FROM Progress)

GROUP BY gr.Name_group

HAVING COUNT (DISTINCT (pr.ID_Lesson))>3

Результат выполнения запроса приведен на рис. 23.

	Группа	Количество_экзаменов
1	07ВП1	8

Рис. 23. Предложение *HAVING* в операторе *SELECT*

Пример 57. Вывести список групп, в которых средний балл > 3:

SELECT gr.Name_group AS Группа, AVG (pr.Examen) AS Средний балл

FROM Groups gr, Progress pr, Students st

WHERE st.ID_Student= pr.ID_Student AND gr.ID_Group=st.ID_Group

GROUP BY gr.Name_group

HAVING AVG (pr.Examen)>3

Результат выполнения запроса приведен на рис. 24.

	Группа	Средний_балл
1	06ВП1	4
2	07ВП1	4

Рис. 24. Предложение *HAVING* в операторе *SELECT*

Краткие итоги

Команда ***SELECT не изменяет данные в базе данных.***

Формат команды ***SELECT***:

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>]

[ORDER BY имя_столбца [,...n]];

Если обрабатывается несколько таблиц, то (при наличии одноименных полей в разных таблицах) в списке полей используется полная спецификация поля, т.е. ***Имя_таблицы.Имя_поля.***

Порядок выполнения команды ***SELECT*** :

1. ***FROM*** – определяются имена используемых таблиц.

2. ***WHERE*** – из указанной таблицы выбираются записи, удовлетворяющие заданным условиям.

Основные типы условий отбора (или предикатов):

– сравнение;

– диапазон;

– принадлежность множеству;

– соответствие шаблону;

– значение ***NULL***.

3. ***GROUP BY*** – выполняется группировка полученных записей, т.е. образуются *группы строк*, имеющих одно и то же значение в указанном столбце.

4. ***HAVING*** – выбор группы строк, удовлетворяющих указанным условиям.

5. ***ORDER BY*** – выполняется сортировка записей в указанном порядке.

6. ***SELECT*** – устанавливается, какие столбцы должны присутствовать в выходных данных.

ORDER BY сортирует данные выходного набора в заданной последовательности. По умолчанию выполняется сортировка по возрастанию, она задается ключевым словом ***ASC.*** Для выполнения сортировки в обратной последовательности необходимо указать ключевое слово ***DESC***. Фраза ***ORDER BY*** всегда должна быть последним элементом в операторе ***SELECT***.

С помощью итоговых (агрегатных) функций можно получить ряд обобщающих статистических сведений о множестве отобранных значений выходного набора.

Основные итоговые 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 определенном поле записей, отобранных запросом. Оно является арифметическим средним значением, т.е. суммой значений, деленной на их количество.

– ***Sum (Выражение)*** – вычисляет сумму множества значений, содержащихся в определенном поле записей, отобранных запросом.

Группирование данных – это размещение данных в столбцах с повторяющимися значениями в определенном логическом порядке.

Запрос, в котором присутствует ***GROUP BY***, называется ***группирующим запросом***. В нем группируются данные, полученные в результате выполнения команды ***SELECT***, после чего для каждой отдельной группы создается единственная суммарная строка.

Предложение ***HAVING*** работает следующим образом:

1. ***GROUP BY*** разделяет строки на наборы (по типу).

2. на полученные группы накладываются условия предложения ***HAVING***.

Контрольные вопросы

1. Как исключить повторяющиеся при выводе записи?

2. Как получить список ***n*** первых (или последних) записей в списке вывода?

3. Какие символы-заменители могут использоваться в шаблоне?

4. Как задать сортировку по нескольким полям?

5. Какие итоговые функции могут использоваться в запросах?

6. В каких случаях используется группировка выводимых записей?

10. Подзапросы

Часто невозможно решить поставленную задачу путем одного запроса. Это особенно актуально, когда при использовании условия поиска в предложении ***WHERE*** значение, с которым надо сравнивать, заранее не определено и должно быть вычислено в момент выполнения команды ***SELECT***. В таком случае используют законченные команды ***SELECT***, внедренные в тело другой команды ***SELECT***.

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

Базовый синтаксис команды с подзапросом:

SELECT имя_столбиа

FROM таблица

WHERE имя_столбца = (SELECT имя__столбца

FROM таблица

WHERE условия);

Подзапрос можно использовать в выражении ключевых слов ***WHERE*** или ***HAVING*** внешних операторов выбора ***SELECT***, вставки ***INSERT,*** обновления ***UPDATE*** или удаления ***DELETE***. Можно использовать логические операции и операции сравнения типа **=, >, <, *IN, NOT IN, AND, OR*** и т.п.

10.1. Правила составления подзапросов

1. Во фразах ***WHERE*** и ***HAVING*** подзапрос записывается как второй операнд условия отбора, т.е. после знака операции сравнения (=, <, >, <=, >=, <>).

2. Текст подзапроса заключается в круглые скобки. Подзапрос может ссылаться только на один столбец в выражении своего ключевого слова ***SELECT.*** Исключение – это случаи, когда в главном запросе используется сравнение с несколькими столбцами из подзапроса.

3. Ключевое слово ***ORDER BY*** нельзя использовать в подзапросе, а только во внешнем подзапросе. Вместо ***ORDER BY*** в подзапросе можно использовать ***GROUP BY***.

4. Подзапрос, возвращающий несколько строк данных, можно использовать только в операторах, допускающих множество значений, например в ***IN.***

5. Подзапрос нельзя использовать как аргумент функции, допускающей множество значений.

6. Подзапросы ***нельзя*** использовать в списках предложений ***ORDER BY*** и ***GROUP BY***.

7. Список выбора внутреннего подзапроса, которому предшествует операция сравнения, может содержать только одно выражение или название столбца, и подзапрос должен возвращать единственный результат. При этом тип данных столбца, указанного в конструкции ***WHERE*** внешнего оператора, должен быть совместим с типом данных в столбце, указанным в списке выбора подзапроса.

8. В подзапросах не допускаются текстовые (text) и графические (image) данные.

9. Подзапросы не могут обрабатывать свои результаты внутренним образом, т.е. подзапрос не может содержать конструкций ***ORDER BY*** или ключевого слова ***INTO***.

10. Количество вложенных уровней для подзапросов не должно превышать 16.

11. Операцию ***BETWEEN*** нельзя использовать по отношению к подзапросу, но ее можно использовать в самом подзапросе.

Пример ***правильного*** использования ***BETWEEN***:

SELECT имя_столбца

FROM таблица

WHERE имя_столбца ОПЕРАЦИЯ (SELECT имя_столбца

FROM таблица

WHERE значение BETWEEN значение);

Пример ***неправильного*** использования ***BETWEEN***:

SELECT имя_столбца FROM таблица

WHERE имя_столбца BETWEEN значение AND (SELECT имя_столбца

FROM таблица);

12. В предложении ***SELECT*** подзапроса нельзя использовать символ “*” и константы (кроме ***EXISTS***-подзапросов).

13. Имена столбцов в подзапросе относятся по умолчанию к таблице, указанной во фразе ***FROM*** подзапроса. Если они относятся к таблице внешнего запроса, необходимо задать полное (уточненное) имя столбца в виде ***“таблица.столбец”***.

14. Список в предложении ***SELECT*** состоит из имен отдельных столбцов или составленных из них выражений – за исключением случая, когда в подзапросе присутствует ключевое слово ***EXISTS***;

10.2. Типы подзапросов

Существуют два основных типа подзапросов:

– подзапросы-выражения или скалярные подзапросы. Этим подзапросам предшествует немодифицированная операция сравнения. Они возвращают единственное значение;

– квантифицированные предикатные подзапросы. Это подзапросы, которые возвращают список значений и которым может предшествовать:

- ключевое слово ***IN*** (принадлежит) или
- операция сравнения, модифицированная кванторами ***ANY*** (некоторый) или ***ALL*** (все).

К этой же группе подзапросов относятся подзапросы, проверяющие существование с помощью квантора ***EXISTS*** (существует).

10.2.1. Скалярный подзапрос

Скалярный подзапрос – это запрос, возвращающий единственное значение. Скалярные подзапросы начинаются с немодифицированного оператора сравнения =, <>, >, >=, <, или <=

В идеале для использования подзапроса с немодифицированной операцией сравнения пользователь должен достаточно хорошо знать табличные данные и понимать характер задачи, чтобы быть уверенным, что подзапрос выдаст единственное значение.

Пример 58. Вывести название предмета, на изучение которого отводится максимальное количество часов:

```
SELECT Nazvanie, Kol_chas
 FROM Lessons
WHERE Kol_chas =
 (SELECT MAX (Kol_chas) FROM Lessons);
```

Во вложенном подзапросе определяется максимальное количество часов, которое отводится на изучение предмета. Во внешнем подзапросе – название предмета, для которого количество часов оказалось равным максимальному.

Результат выполнения запроса приведен на рис. 25.

	Nazvanie	Kol_chas
1	Теория вероятности	102

Рис. 25. Скалярный подзапрос

Следует отметить, что ***нельзя прямо использовать предложение WHERE Количество = MAX (Количество), поскольку применять обобщающие функции в предложениях WHERE запрещено.*** Для достижения желаемого результата следует создать подзапрос, вычисляющий максимальное значение количества, а затем использовать его во внешнем операторе ***SELECT***, предназначенном для выборки названия предмета, где количество часов совпало с максимальным значением.

Пример 59. Вывести список студентов, получающих стипендию выше средней, и указать для этих студентов превышение над средним уровнем:

SELECT FIO AS ФИО, Stipendiya AS Стипендия,
Stipendiya - (SELECT AVG (Stipendiya)
FROM Students)
AS Превышение FROM Students
WHERE Stipendiya > (SELECT AVG (Stipendiya)
FROM Students);

Результат выполнения запроса приведен на рис. 26.

	ФИО	Стипендия	Превышение
1	Иванков С.В.	2000	430
2	Карпов А.В.	1700	130
3	Ивкин И.Ю.	1700	130
4	Горин А.А.	1700	130
5	Коряшкин А.С.	1700	130

Рис. 26. Скалярный подзапрос
с немодифицированным оператором сравнения

В подзапросе вычисляется среднее значение стипендии. Во внешнем запросе ***SELECT*** используется как для вычисления отклонения количества от среднего уровня, так и для отбора сведений о ФИО.

Пример 60. Определить предметы, по которым средний балл больше 3:

SELECT ls.Nazvanie AS Название предмета, AVG (pr.Examen) AS Средний балл
FROM Lessons ls, Progress pr
WHERE pr.ID_Lesson=ls.ID_Lesson
GROUP BY ls.Nazvanie
HAVING AVG (pr.Examen) > 3;

Результат выполнения запроса приведен на рис. 27.

	Предмет	Средний_балл
1	Базы данных	5
2	Математическая логика и теория алгоритмов	4
3	Объектно-ориентированное программирование	4
4	Операционные системы	5
5	Организация ЭВМ	4
6	Человеко-машинное взаимодействие	5

Рис. 27. Вспомогательный запрос для примера 61

Пример 61. Определить предметы, по которым средний балл оказался больше среднего балла по всем предметам вообще.

К запросу, приведенному в примере 60, добавим подзапрос:

SELECT ls.Nazvanie AS Название предмета, AVG(pr.Examen) AS Средний балл
FROM Lessons ls, Progress pr
WHERE pr.ID_Lesson=ls.ID_Lesson
GROUP BY ls.Nazvanie
HAVING AVG (pr.Examen) > (SELECT AVG (examen)
FROM Progress);

Результат выполнения запроса приведен на рис. 28.

Results | Messages

	Предмет	Средний_балл
1	Базы данных	5
2	Операционные системы	5
3	Человеко-машинное взаимодействие	5

Рис. 28. *SELECT* с *GROUP BY* и *HAVING*

Пример 62. Найти номер группы, в которой куратор – преподаватель Мещеряков Б. К.:

SELECT Name_group AS Название группы
FROM Groups
WHERE Kurator=(SELECT ID_Teacher
FROM Teachers
WHERE Familia='Мещеряков');

Результат выполнения запроса приведен на рис. 29.

	Номер_группы	Название_группы
1	16	08ВП1

Рис. 29. *SELECT* со связанным подзапросом

Порядок выполнения оператора ***SELECT*** со связанным подзапросом:

1. Выбирается строка из таблицы, имя которой указано во внешнем запросе.
2. Выполняется подзапрос и полученное значение применяется для анализа этой строки в условии предложения ***WHERE*** внешнего запроса.
3. По результату оценки этого условия принимается решение о включении или невключении строки в состав выходных данных.
4. Процедура повторяется для следующей строки таблицы внешнего запроса.

Следует обратить внимание, что приведенный выше запрос корректен только в том случае, если в результате выполнения указанного в скобках подзапроса возвращается единственное значение. Если в результате выполнения подзапроса будет возвращено несколько значений, то этот подзапрос будет ошибочным. В данном примере это произойдет, если в таблице ***Groups*** будет несколько записей со значениями поля ***Familia*** = ***'Мещеряков'***.

10.2.2. Подзапросы, возвращающие множество значений

Во многих случаях значение, подлежащее сравнению в предложениях ***WHERE*** или ***HAVING***, представляет собой не одно, а несколько значений. Вложенные подзапросы генерируют непоимено-

ванное промежуточное отношение, временную таблицу. Оно может использоваться только в том месте, где появляется в подзапросе. К такому отношению невозможно обратиться по имени из какого-либо другого места запроса.

Применяемые к подзапросу операции основаны на операциях, которые применяются к множеству, а именно:

– ***{ WHERE | HAVING } выражение [NOT] IN (подзапрос);***

– ***{ WHERE | HAVING } выражение оператор_сравнения { ALL | SOME | ANY} (подзапрос);***

– ***{WHERE | HAVING } [NOT] EXISTS (подзапрос);***

Использование операций IN и NOT IN

Предикат ***IN*** используется для отбора в главном запросе только тех записей, которые содержат значения, совпадающие с одним из отобранных подчиненным запросом:

IN – равно любому из значений, полученных во внутреннем запросе.

NOT IN – не равно ни одному из значений, полученных во внутреннем запросе.

Пример 63. Выбрать студентов, которые получили на экзамене оценку не менее 4:

SELECT * FROM Students

WHERE ID_Student IN

(SELECT ID_Student FROM Progress

WHERE Examen >= 4);

Результат выполнения запроса приведен на рис. 30.

	ФИО	Номер зачетной книжки
1	Иванков С.В.	08ВП129
2	Илюхин В.И.	08ВП103
3	Карпов А.В.	06ВП111
4	Ивкин И.Ю.	06ВП110
5	Заваровский К.Ю.	06ВП109

Рис. 30. Использование предиката *IN*

Предикат ***NOT IN*** используется для отбора в главном запросе только тех записей, которые содержат значения, не совпадающие ни с одним из отобранных подчиненным запросом.

Введение в запрос фразы «***только***» требует использования операции ***NOT IN***.

Пример 64. Вывести фамилии студентов, которые получили на экзаменах оценки менее 4:

SELECT FIO FROM Students

WHERE ID_Student NOT IN

(SELECT ID_Student FROM Progress

WHERE Examen >= 4);

Результат выполнения запроса приведен на рис. 31.

	FIO
1	Макарь В.А.
2	Ивкин И.Ю.
3	Заваровский К.В.
4	Болтоносов И.Ю.
5	Горин А.А.
6	Додонов А.С.
7	Коряшкин А.С.
8	Янин А.В.
9	Бирюков В.В.
10	Иваненков С.В.

Рис. 31. Использование предиката *NOT IN*

Использование ключевого слова ANY

Ключевые слова ***ALL*** и ***ANY*** модифицируют операцию сравнения, которая формирует подзапрос. Ключевые слова ***ANY*** и ***ALL*** могут использоваться с подзапросами, возвращающими один столбец чисел.

ANY (SOME) – оператор, эквивалентный понятию «***любой***».

Выражение **=** ***ANY(…)*** означает: равно одному из значений элементов результирующего множества; эквивалентно использованию предиката ***IN***. Соответствует логическому оператору ***OR***.

Выражение <> ***ANY(…)*** не эквивалентно ***NOT IN***: оно выполняется всегда, кроме случаев ***NULL***-значений.

Выражение > ***ANY*** означает: больше, чем любое полученное число; эквивалентно операции > для самого меньшего полученного

числа. Другими словами, ***>ANY*** означает больше, по крайней мере, одного значения или (что равносильно) больше минимальной величины. Поэтому > ***ANY (1,2,3)*** означает больше 1.

Выражение > = ***ANY*** означает: больше или равно любому полученному числу; эквивалентно операции > = для самого меньшего полученного числа.

Выражение < ***ANY*** означает: меньше, чем любое полученное число; эквивалентно < для самого большего полученного числа.

Выражение < = ***ANY*** означает: меньше или равно любому полученному числу; эквивалентно операции < = для самого большего полученного числа.

Пример 65. Вывести идентификаторы студентов, у которых оценки превосходят величину, по крайней мере, одной из оценок, полученных ими же в 6 семестре.

```
SELECT DISTINCT ID_Student AS [Номер студента]
FROM Progress
WHERE Examen >ANY
(SELECT Examen
FROM Progress
WHERE Semestr = 6);
```

Результат выполнения запроса приведен на рис. 32.

	Номер студента
1	74
2	79
3	81
4	83
5	84

Рис. 32. Использование ключевого слова *ANY*

Использование ключевого слова ALL

ALL – это оператор, эквивалентный понятию "***все***".

Выражение = ***ALL*** означает: равно всем полученным значениям; эквивалентно логическому оператору ***AND***.

Выражение > ***ALL*** означает: больше, чем все полученные числа, или (что равносильно) больше максимальной величины. Напри-

мер, > ***ALL (1,2,3)*** означает больше, чем 3. Это эквивалент > для самого большего полученного числа.

Выражение > = ***ALL*** означает: больше или равно всем полученным числам. Это эквивалент >= для самого большего полученного числа.

Если подзапросу будет предшествовать ключевое слово ***ALL***, условие сравнения считается выполненным, только когда оно выполняется для ***всех*** значений в результирующем столбце подзапроса.

Если записи подзапроса предшествует ключевое слово ***ANY***, то условие сравнения считается выполненным, когда оно выполняется хотя бы для о***дного*** из значений в результирующем столбце подзапроса.

Если в результате выполнения подзапроса получено пустое значение, то для ключевого слова ***ALL*** условие сравнения будет считаться выполненным, а для ключевого слова ***ANY*** – невыполненным.

Ключевое слово ***SOME*** является синонимом слова ***ANY***.

Предикат ***ALL*** используется для отбора в главном запросе только тех записей, которые удовлетворяют сравнению со всеми записями, отобранными в подчиненном запросе. Если в примере 65 предикат ***ANY*** заменить предикатом ***ALL***, результат запроса будет включать только тех студентов, у которых оценки превосходят все оценки, полученные ими же в 6 семестре. Это условие является значительно ***более жестким.***

Пример 66. Вывести данные о сдаче экзаменов с оценками меньше, чем все полученные оценки в 5 семестре.

```
SELECT ID_Student AS [Номер студента],
ID_Lesson [Номер предмета], Semestr AS [Ctvtcnh],
Examen AS [Экзаменационная оценка]
FROM Progress
WHERE Examen < ALL
(SELECT Examen
FROM Progress
WHERE Semestr = 5) ;
```

Результат выполнения запроса приведен на рис. 33.

	Номер студента	Номер предмета	Семестр	Экзаменационная оценка
1	75	1	4	3
2	84	5	7	3

Рис. 33. Использование ключевого слова *ALL*

Использование ключевых слов EXISTS и NOT EXISTS

Предикат ***EXISTS*** (***NOT EXISTS***) – квантор существования, используется в логическом выражении для определения того, должен ли подчиненный запрос возвращать какие-либо записи.

В языке Transact SQL предикат с квантором существования представляется выражением ***EXISTS (SELECT * FROM ...).***

Ключевые слова ***EXISTS*** и ***NOT EXISTS*** предназначены для использования только совместно с ***подзапросами***. Результат их обработки представляет собой логическое значение ***TRUE*** или ***FALSE***.

Для ключевого слова ***EXISTS*** результат равен ***TRUE*** в том и только в том случае, если в возвращаемой подзапросом результирующей таблице присутствует хотя бы одна строка.

Если результирующая таблица подзапроса пуста, результатом обработки операции ***EXISTS*** будет значение ***FALSE***.

Для ключевого слова ***NOT EXISTS*** используются правила обработки, обратные по отношению к ключевому слову ***EXISTS***.

Поскольку по ключевым словам ***EXISTS*** и ***NOT EXISTS*** проверяется лишь наличие строк в результирующей таблице подзапроса, то эта таблица может содержать произвольное количество столбцов.

Пример 67. Вывести список студентов, которые сдавали экзамены.

```
SELECT Fio
FROM Students
WHERE EXISTS (SELECT ID_Student
 FROM Progress
WHERE Students.ID_Student=Progress.ID_Student);
```

Результат выполнения запроса приведен на рис. 34.

	FIO
1	Иванков С.В.
2	Буртасов И.Ю.
3	Глинин И.В.
4	Илюхин В.И.
5	Карпов А.В.
6	Ивкин И.Ю.
7	Заваровский К.Ю.

Рис. 34. Использование ключевого слова *EXISTS*

Пример 68. Вывести список студентов, которые не сдавали экзамены.

SELECT FIO
FROM Students
WHERE NOT EXISTS (SELECT ID_Student
FROM Progress
WHERE Students.ID_Student=Progress.ID_Student);

EXISTS представляет собой одну из наиболее важных возможностей Transact SQL. Фактически любой запрос, который выражается через ***IN***, может быть альтернативным образом сформулирован также с помощью ***EXISTS***. Однако обратное высказывание несправедливо.

Ключевое слово ***EXISTS*** является очень важным, поскольку часто не существует альтернативного способа выбора данных без использования подзапроса. Подзапросы, которым предшествует квантор ***EXISTS,*** всегда являются ***коррелированными***.

Подзапрос, которому предшествует квантор существования ***EXISTS***, имеет по сравнению с другими подзапросами следующие особенности:

– перед ключевым словом ***EXISTS*** не должно быть названий столбцов, констант или других выражений;

– подзапрос с квантором существования возвращает значения ***TRUE*** или ***FALSE*** и не возвращает никаких данных из таблицы;

– список выбора такого подзапроса часто состоит из одной звездочки (*). Здесь нет необходимости указывать названия столб-

цов, поскольку осуществляется просто проверка существования строк, удовлетворяющих условиям, указанным в подзапросе. Здесь можно и явно указать список выбора, следуя обычным правилам.

10.3. Виды вложенных подзапросов

Подзапросы любого из рассмотренных выше типов могут быть либо коррелированными (повторяющимися), либо некоррелированными.

Некоррелированный подзапрос (***простой вложенный подзапрос)*** может вычисляться как независимый запрос. Иначе говоря, результаты подзапроса подставляются в основной оператор (или внешний запрос). Это не значит, что SQL-сервер именно так выполняет операторы с подзапросами. Некоррелированные подзапросы могут быть заменены соединением и будут выполняться как соединения SQL-сервером.

Коррелированный подзапрос не может выполняться как независимый запрос, поскольку он содержит условия, зависящие от значений полей в основном запросе. Запросы на существование обычно являются коррелированными.

Простые вложенные подзапросы обрабатываются системой "снизу вверх". Первым обрабатывается вложенный подзапрос самого нижнего уровня. Множество значений, полученное в результате его выполнения, используется при реализации подзапроса более высокого уровня и т.д.

Запросы с коррелированными вложенными подзапросами обрабатываются системой в обратном порядке. Сначала выбирается первая строка рабочей таблицы, сформированной основным запросом, и из нее выбираются значения тех столбцов, которые используются во вложенном подзапросе (вложенных подзапросах). Если эти значения удовлетворяют условиям вложенного подзапроса, то выбранная строка включается в результат. Затем выбирается вторая строка и т.д., пока в результат не будут включены все строки, удовлетворяющие вложенному подзапросу (последовательности вложенных подзапросов).

10.3.1. Коррелированные вложенные подзапросы

Пример 69. Вывести группы, в которых учатся студенты, получающие стипендию в размере 2000 рублей:

SELECT Name_group AS Группа
FROM Groups
WHERE 2000 IN
(SELECT Stipendiya
FROM Students
WHERE Groups.ID_Group = Students.ID_Group);

Результат выполнения запроса приведен на рис. 35.

Группа
06ВП1

Рис. 35. Коррелированный вложенный подзапрос

Подобные подзапросы называются ***коррелированными***, так как их результат зависит от значений, определенных во внешнем подзапросе. Следовательно**,** обработка коррелированного подзапроса должна повторяться для каждого значения, извлекаемого из внешнего подзапроса, а не выполняться раз и навсегда.

Пример 70. Выбрать сведения обо всех предметах обучения, по которым были получены оценки ‘5’:

SELECT DISTINCT Nazvanie AS [Название дисциплины],
Kol_chas AS [Количество часов]
FROM Lessons SO
WHERE 5 IN
(SELECT Examen
FROM Progress EX
WHERE SO.ID_Lesson = EX.ID_Lesson)

В приведенном запросе ***SO*** и ***EX*** являются псевдонимами (алиасами), т.е. специально вводимыми именами, которые могут быть использованы в данном запросе вместо настоящих имен. В приведенном примере они используются вместо имен таблиц ***Lessons*** и ***Progress***.

Результат выполнения запроса приведен на рис. 36.

	Название дисциплины	Количество часов
1	Объектно-ориентированное программирование	68
2	Операционные системы	51
3	Организация ЭВМ	51
4	Человеко-машинное взаимодействие	51
5	Базы данных	51

Рис. 36. Коррелированный вложенный подзапрос

Пример 71. Задачу из примера 70 можно решить с помощью операции соединения таблиц:

SELECT DISTINCT

Lessons.ID_Lesson, Lessons.Nazvanie, Lessons.Kol_chas, Progress.Semestr

FROM Lessons , Progress

WHERE Lessons. ID_Lesson = Progress. ID_Lesson

AND Progress. Examen = 5;

Результат выполнения запроса приведен на рис. 37.

	Название дисциплины	Количество часов	Семес...
1	Базы данных	51	6
2	Объектно-ориентированное программирование	68	4
3	Операционные системы	51	6
4	Организация ЭВМ	51	5
5	Человеко-машинное взаимодействие	51	6

Рис. 37. Операции соединения таблиц

Можно использовать подзапросы, связывающие таблицу со своей собственной копией.

Пример 72. Найти фамилии и стипендии студентов, получающих стипендию выше средней стипендии в группе, в которой они учатся:

SELECT Fio AS [ФИО], Stipendiya AS [Стипендия]

FROM Students El

WHERE Stipendiya >

(SELECT AVG(Stipendiya)

FROM Students E2

WHERE El.ID_Group = E2.ID_Group);

Результат выполнения запроса приведен на рис. 38.

	ФИО	Стипендия
1	Горин А.А.	1700
2	Коряшкин А.С.	1700
3	Иванков С.В.	2000
4	Карпов А.В.	1700
5	Ивкин И.Ю.	1700

Рис. 38. Подзапрос, связывающий таблицу со своей копией

10.3.2. Связанные подзапросы в HAVING

GROUP BY позволяет группировать выводимые ***SELECT***-запросом записи по значению некоторого поля. Использование предложения ***HAVING*** позволяет при выводе осуществлять фильтрацию таких групп.

Предикат предложения ***HAVING*** оценивается не для каждой строки результата, а для каждой группы выходных записей, сформированной предложением ***GROUP BY*** внешнего запроса.

Пример 73. Определить среднюю из полученных студентами оценок, сгруппировав значения оценок по семестрам и исключив те семестры, когда число студентов, сдававших в течение дня экзамены, было меньше 10:

SELECT Semestr AS Семестр, Avg(Examen) AS [Средняя оценка на экзамене]

FROM Progress A

GROUP BY Semestr

HAVING 10 <

(SELECT COUNT(Examen)

FROM Progress B

WHERE A.Semestr = B.Semestr);

Результат выполнения запроса приведен на рис. 39.

	Семестр	Средняя оценка на экзамене
1	4	4
2	5	4
3	6	4

Рис. 39. Связанные подзапросы в *HAVING*

Подзапрос вычисляет количество строк с одним и тем же семестром, совпадающим с семестром, для которого сформирована очередная группа основного запроса.

10.4. Подзапросы в командах модификации

10.4.1. Подзапросы в команде INSERT

Подзапросы могут использоваться и с командами языка манипулирования данными (DML).

Команда ***INSERT*** использует данные, возвращаемые подзапросом, для помещения их в другую таблицу. Выбранные в подзапросе данные можно модифицировать с помощью символьных или числовых функций, а также функций дат и времени.

Базовый синтаксис соответствующей команды должен быть следующим.

INSERT INTO имя_таблицы [(столбец1 [, столбец2])]

SELECT [*| столбец1 [, столбец2]]

FROM таблица1 [, таблица2]

[WHERE значение ОПЕРАЦИЯ значение];

Вот пример использования команды ***INSERT*** с подзапросом.

Пример 74. Пусть таблица ***STUDENTI*** имеет структуру, полностью совпадающую со структурой таблицы ***STUDENT***. Запрос, позволяющий заполнить таблицу ***STUDENTI*** записями из таблицы ***STUDENT*** обо всех студентах, получающих стипендию в размере 1700 рублей, выглядит следующим образом:

INSERT INTO Students1

SELECT *

FROM Students

WHERE Stipendiya = 1700;

10.4.2. Подзапросы в команде UPDATE

С помощью команды ***UPDATE*** с подзапросом можно обновлять данные как одного, так и нескольких столбцов сразу.

Базовый синтаксис команды следующий:

UPDATE таблица

SET имя_столбца [, имя_столбца] =

(SELECT имя_столбца [,имя_столбца] FROM таблица
[WHERE]);

Рассмотрим примеры, поясняющие использование команды ***UPDATE*** с подзапросом.

Пример 75. Увеличить значение размера стипендии на 200 руб. в записях студентов, сдавших экзамены на 4 и 5:

UPDATE Students
SET Stipendiya = Stipendiya + 200
WHERE 4 <=
(SELECT MIN(Examen)
FROM Progress
WHERE Progress.ID_Student = Students.ID_Student);

Пример 76. Уменьшить величину стипендии на 200 руб. всем студентам, получившим на экзамене минимальную оценку:

UPDATE Students
SET Stipendiya = Stipendiya - 200
WHERE ID_Student IN
(SELECT ID_Student
FROM Progress A
WHERE Examen =
(SELECT MIN(Examen)
FROM Progress B
WHERE A.Semestr = B. Semestr));

10.4.3. Подзапросы в команде DELETE

Базовый синтаксис команды следующий.

DELETE FROM имя_таблицы
[WHERE ОПЕРАЦИЯ [значение]
(SELECT имя_столбца
FROM имя_таблицы
[WHERE]);

Пример 77. Найти наименьшее значение оценки, полученной в каждом семестре, и удалить из таблицы сведения о студенте, который получил эту оценку. Запрос будет иметь вид:

```
DELETE
 FROM Progress
 WHERE ID_Student IN
 (SELECT DISTINCT ID_Student
 FROM Progress A
 WHERE Examen=
 (SELECT MIN(Examen)
 FROM Progress B
 WHERE A.Semestr = B .Semestr));
```

Краткие итоги

Часто невозможно решить поставленную задачу путем одного запроса. Это особенно актуально, когда при использовании условия поиска в предложении ***WHERE*** значение, с которым надо сравнивать, заранее не определено и должно быть вычислено в момент выполнения команды ***SELECT***. В таком случае используют команды ***SELECT***, внедренные в тело другой команды ***SELECT***.

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

Базовый синтаксис оператора с подзапросом:

```
SELECT имя_столбиа
FROM таблица
WHERE имя_столбца = (SELECT имя__столбца
FROM таблица
WHERE условия);
```

Подзапрос можно использовать в выражении ключевых слов ***WHERE*** или ***HAVING*** внешних команд выбора ***SELECT***, вставки ***INSERT,*** обновления ***UPDATE*** или удаления ***DELETE***. Можно использовать логические операции и операции сравнения типа =, >, <, ***IN, NOT IN, AND, OR*** и т.п.

Существуют два основных типа подзапросов:

– скалярные подзапросы**.** Этим подзапросам предшествует немодифицированная операция сравнения. Они возвращают единственное значение;

– квантифицированные предикатные подзапросы. Это подзапросы, которые возвращают список значений и которым может предшествовать ключевое слово ***IN*** (принадлежит) или операция сравнения, модифицированная кванторами ***ANY*** (некоторый) или ***ALL*** (все).

К этой же группе подзапросов относятся подзапросы, проверяющие существование с помощью квантора ***EXISTS*** (существует).

Подзапросы также могут быть либо коррелированными (повторяющимися), либо некоррелированными.

Некоррелированный подзапрос (простой вложенный подзапрос) может вычисляться как независимый запрос.

Коррелированный подзапрос не может выполняться как независимый запрос, поскольку он содержит условия, зависящие от значений полей в основном запросе. Запросы на существование обычно являются коррелированными.

Простые вложенные подзапросы обрабатываются системой «снизу вверхШПервым обрабатывается вложенный подзапрос самого нижнего уровня. Множество значений, полученное в результате его выполнения, используется при реализации подзапроса более высокого уровня и т.д.

Запросы с коррелированными вложенными подзапросами обрабатываются системой в обратном порядке. Сначала выбирается первая строка рабочей таблицы, сформированной основным запросом, и из нее выбираются значения тех столбцов, которые используются во вложенном подзапросе (вложенных подзапросах). Если эти значения удовлетворяют условиям вложенного подзапроса, то выбранная строка включается в результат. Затем выбирается вторая строка и т.д., пока в результат не будут включены все строки, удовлетворяющие вложенному подзапросу (последовательности вложенных подзапросов).

Подзапросы можно использовать в командах модификации.

Команда ***INSERT*** использует данные, возвращаемые подзапросом, для помещения их в другую таблицу.

С помощью команды ***UPDATE*** с подзапросом можно обновлять данные как одного, так и нескольких столбцов сразу.

Подзапросы можно использовать также в команде ***DELETE*** для удаления записей, найденных с помощью команды ***SELECT.***

Контрольные вопросы

1. Что такое подзапрос?
2. Почему возникает необходимость создания подзапросов?
3. Каковы правила составления подзапросов?
4. Что такое скалярный подзапрос?
5. Какие подзапросы возвращают множество значений?
6. В чем различие при выполнении коррелированных и некоррелированных запросов?
7. Для каких целей используются команды ***INSERT INTO***, ***DELETE FROM*** и ***UPDATE*** с подзапросами?

11. Команда UNION

Команда ***UNION*** используется для объединения результатов двух или более команд ***SELECT*** с исключением повторяющихся строк. Другими словами, если строка попадает в вывод одного запроса, то второй раз она не выводится, даже если она возвращается вторым запросом. При использовании ***UNION*** в каждом из связываемых команд ***SELECT*** выполняются следующие условия:

1) должно быть выбрано одинаковое число столбцов;

2) столбцы должны быть одинакового типа и следовать в том же порядке.

Синтаксис команды:

SELECT столбец1 [,… столбецN]
FROM таблица1 [,… таблицаM]
[WHERE]
UNION
SELECT столбец1 [, …столбецN]
FROM таблица1 [, …таблицаM]
[WHERE];

Пример 78. Для того, чтобы получить в одной таблице фамилии и идентификаторы студентов и преподавателей кафедры МОиПЭВМ, можно использовать следующий запрос:

SELECT 'Студент' AS 'Студент/преподаватель',
Fio AS 'ФИО' FROM Students WHERE ID_Kaf=1

UNION

SELECT 'Преподаватель' AS 'Студент/преподаватель', Familia AS 'ФИО' FROM Teachers WHERE ID_Kaf=1;

Результат выполнения запроса приведен на рис. 40.

	Студент/преподаватель	ФИО
1	Преподаватель	Гурьянов
2	Преподаватель	Мещеряков
3	Преподаватель	Самуйлов
4	Преподаватель	Сивохин
5	Преподаватель	Шашков
6	Преподаватель	Шибанов
7	Студент	Абышкин В.В.
8	Студент	Бирюков В.В.
9	Студент	Болтоносов И.Ю.
10	Студент	Буртасов И.Ю.

Рис. 40. Объединения результатов двух операторов *SELECT*

Пример 79. Для получения в одной таблице ФИО студента, название дисциплины и оценку за 6 семестр, а также дисциплины, по которым оценка «отлично», необходимо использовать следующий запрос:

SELECT Fio AS ’ФИО’, Nazvanie AS ’Дисциплина’, Examen AS ’Оценка’

FROM Students, Progress, Lessons

WHERE Students.ID_Student=Progress.ID_Student AND

Lessons.ID_Lesson=Progress.ID_Lesson AND

Semestr=6

UNION

SELECT Fio, Nazvanie, Examen

FROM Students, Progress, Lessons

WHERE Students.ID_Student=Progress.ID_Student AND

Lessons.ID_Lesson=Progress.ID_Lesson AND

Examen=5;

Результат выполнения запроса приведен на рис. 41.

	ФИО	Дисциплина	Оценка
1	Абышкин В.В.	Математическая логика и теория алгоритмов	4
2	Абышкин В.В.	Объектно-ориентированное программирование	5
3	Абышкин В.В.	Операционные системы	5
4	Заваровский К.Ю.	Объектно-ориентированное программирование	5
5	Заваровский К.Ю.	Операционные системы	5
6	Иванчуков А.Г.	Базы данных	5
7	Иванчуков А.Г.	Математическая логика и теория алгоритмов	4

Рис. 41. Объединения результатов двух команд *SELECT*

Использование команды ***UNION*** возможно только при объединении запросов, соответствующие столбцы которых совместимы по объединению, т.е.:

– соответствующие числовые поля должны иметь полностью совпадающие тип и размер;

– символьные поля должны иметь точно совпадающее количество символов;

– если ***NULL***-значения запрещены для столбца хотя бы одного любого подзапроса объединения, то они должны быть запрещены и для всех соответствующих столбцов в других подзапросах объединения.

В отличие от обычных запросов ***UNION*** автоматически исключает из выходных данных дубликаты строк.

Пример 80. В этом запросе совпадающие идентификаторы кафедр будут исключены:

SELECT ID_Kaf
FROM students
UNION
SELECT ID_Kaf
FROM Teachers;

Если все же необходимо в каждом запросе вывести все строки независимо от того, имеются ли такие же строки в других объединяемых запросах, то следует использовать во множественном запросе конструкцию с командой ***UNION ALL***.

Пример 81. Дубликаты значений кафедр, выводимые второй частью запроса, не будут исключаться:

SELECT ID_Kaf
FROM students
UNION ALL
SELECT ID_Kaf
FROM Teachers;

Краткие итоги

Команда ***UNION*** используется для объединения результатов двух или более операторов ***SELECT*** с исключением повторяющихся строк.

Использование команды ***UNION*** возможно только при объединении запросов, соответствующие столбцы которых совместимы по объединению, т.е.:

– соответствующие числовые поля должны иметь полностью совпадающие тип и размер;

– символьные поля должны иметь точно совпадающее количество символов;

– если ***NULL***-значения запрещены для столбца хотя бы одного любого подзапроса объединения, то они должны быть запрещены и для всех соответствующих столбцов в других подзапросах объединения.

В отличие от обычных запросов ***UNION*** автоматически исключает из выходных данных дубликаты строк.

Если все же необходимо в каждом запросе вывести все строки независимо от того, имеются ли такие же строки в других объединяемых запросах, то следует использовать во множественном запросе конструкцию с командой ***UNION ALL***.

Контрольные вопросы

1. Для чего используется команда ***UNION***?

2. Какие существуют ограничения на использование команды ***UNION***?

3. Как объединить результаты двух или более команд ***SELECT*** без исключения повторяющихся строк?

12. Соединение таблиц

Для соединения таблиц с одноименными столбцами или таблицы с самой собой используются алиасы или псевдонимы. Они задаются во фразе ***FROM*** через пробел после имени таблицы. При этом истинное имя таблицы в базе данных не меняется.

Например:

SELECT R.a1, R.a2, S.b1, S.b2
FROM R t1, S t2
WHERE R.a1= S.b2;

12.1. Внутреннее соединение (INNER JOIN)

При внутреннем естественном соединении группируются только те строки, значения которых по соединяемым (одноименным) столбцам совпадают:

SELECT R.a1, R.a2, S.b1, S.b2
FROM R, S
WHERE R.a2=S.b1

или

SELECT R.a1, R.a2, S.b1, S.b2
FROM R INNER JOIN S ON R.a2=S.b1;

Пример 82. Объединить поля из таблиц ***Teachers*** и ***Groups*** при условии, что преподаватель является куратором группы:

SELECT Familia, Imja, Surname, Groups.Kurator
FROM Teachers
INNER JOIN Groups ON Teachers.ID_Teacher Groups.Kurator;

Результат выполнения запроса приведен на рис. 42.

	Familia	Name_Group
1	Гурьянов	06ВП2
2	Шашков	06ВП1
3	Сивохин	06ВВ1
4	Шибанов	07ВП1
5	Самуйлов	07ВП2
6	Мещеряков	08ВП1

Рис. 42. Внутреннее соединение

12.2. Внешнее соединение

При внешнем соединении в результирующую таблицу помещаются не только парные строки, но и строки, не нашедшие себе пару. По способу добавления непарных строк различают:

– левое открытое соединение, когда непарные строки добавляются из таблицы, расположенной слева по отношению к опции ***JOIN*** ;

– правое открытое соединение, когда непарные строки добавляются из правой по отношению к ***JOIN*** таблицы;

– полное открытое соединение, когда добавляются все непарные строки обеих соединяемых таблиц.

12.2.1. Внешнее левое соединение LEFT JOIN

При внешнем левом соединении в результирующий набор будут выбраны все строки из левой таблицы (указываемой первой). При совпадении значений по соединяемым (одноименным) столбцам значения второй таблицы заносятся в результирующий набор в соответствующие строки. При отсутствии совпадений в качестве значений второй таблицы проставляется значение ***NULL***:

SELECT R.a1, R.a2, S.b1, S.b2

FROM R LEFT JOIN S ON R.a2=S.b1;

Пример 83. Соединить поле ***Familia*** из таблицы ***Teachers*** с полем ***Name_Group*** из таблицы ***Groups:***

SELECT Teachers.Familia, Groups.Name_Group FROM Teachers

LEFT JOIN Groups ON Teachers.ID_Teacher=Groups.Kurator;

12.2.2. Внешнее правое соединение RIGHT JOIN

При внешнем правом соединении в результирующий набор будут выбраны все строки из правой таблицы (указываемой второй). При совпадении значений по соединяемым (одноименным) столбцам значения первой таблицы заносятся в результирующий набор в соответствующие строки. При отсутствии совпадений в качестве значений первой таблицы проставляется значение ***NULL***:

SELECT R.a1, R.a2, S.b1, S.b2

FROM R RIGHT JOIN S ON R.a2=S.b1;

Пример 84. Объединить таблицы ***Lessons*** и ***Progress***, используя правое соединение по предметам:

SELECT Nazvanie, Examen

FROM Lessons

RIGHT JOIN Progress ON Lessons.ID_Lesson=Progress.ID_Lesson;

Результат выполнения запроса приведен на рис. 43.

	Название дисциплины	Экзам_оценка
1	Объектно-ориентированное программирование	3
2	Объектно-ориентированное программирование	5
3	Операционные системы	5
4	Организация ЭВМ	4
5	Объектно-ориентированное программирование	5
6	Операционные системы	5
7	Организация ЭВМ	4

Рис. 43. Внешнее правое соединение

12.2.3. Полное внешнее соединение FULL JOIN

При полном внешнем соединении в результирующий набор будут выбраны все строки, как из правой, так и из левой таблицы. При совпадении значений по соединяемым (одноименным) столбцам строка содержит значения как из левой, так и из правой таблицы. В противном случае, вместо отсутствующих значений в столбцы таблицы (левой или правой) заносится значение ***NULL***.

Пример 85. Объединить таблицы ***Teacher*** и ***Groups***, используя полное соединение по преподавателям:

SELECT Teachers.ID_Teacher, Familia, Imja, Surname, Groups.Kurator

FROM Teachers FULL JOIN Groups ON Teachers.ID_Teacher=Groups.Kurator;

12.3. Использование псевдонимов при соединении таблиц

Часто при запросе информации необходимо осуществлять соединение таблицы с ее же копией. Например, это требуется в случае, когда нужно найти фамилии студентов, имеющих одинаковые имена. При соединении таблицы с ее же копией вводят псевдонимы (алиасы) таблицы. Запрос для такого случая выглядит следующим образом:

Пример 86. Вывести фамилии преподавателей, имеющих одинаковые имена:

SELECT first.Familia, second.Familia
FROM Teachers first, Teachers second
WHERE first.Imja = second.Imja;

Результат выполнения запроса приведен на рис. 44.

	Familia	Familia
1	Гурьянов	Гурьянов
2	Гурьянов	Гурьянов
3	Гурьянов	Гурьянов
4	Шашков	Шашков
5	Мещеряков	Шашков
6	Сивохин	Сивохин
7	Шибанов	Шибанов
8	Самуйлов	Шибанов
9	Шибанов	Самуйлов

Рис. 44. Использование псевдонимов при соединении таблиц

В этом запросе введены два псевдонима для одной таблицы ***Teachers***, что позволяет корректно задать выражение, связывающее две копии таблицы. Чтобы исключить повторения строк в выводимом результате запроса из-за повторного сравнения одной и той же пары преподавателей, необходимо задать порядок следования для двух значений так, чтобы одно значение было меньше, чем другое, что делает предикат асимметричным.

Пример 87. Задать порядок следования

SELECT first.Familia, second.Familia
FROM Teachers first, Teachers second
WHERE first.Imja = second.Imja AND
first.Familia < second.Familia;

Результат выполнения запроса приведен на рис. 45.

	Familia	Familia
1	Мещеряков	Шашков
2	Самуйлов	Шибанов

Рис. 45. Исключение повторения строк

12.4. Операции соединения таблиц посредством ссылочной целостности

Информация в таблицах ***Student*** и ***Progress*** уже связана посредством поля ***ID_Student***. В таблице ***Student*** поле ***ID_Student*** является первичным ключом, а в таблице ***Progress*** – ссылающимся на него внешним ключом. Состояние связанных таким образом таблиц называется ***состоянием ссылочной целостности***. В данном случае ссылочная целостность этих таблиц подразумевает, что каждому значению поля ***ID_Student*** в таблице ***Progress*** обязательно соответствует такое же значение поля ***ID_Student*** в таблице ***Student***. Другими словами, в таблице ***Progress*** не может быть записей, имеющих идентификаторы студентов, которых нет в таблице ***Student***. Стандартное применение операции соединения состоит в извлечении данных в терминах этой связи.

Пример 88. Получить список фамилий студентов с полученными ими оценками и идентификаторами предметов:

SELECT FIO AS 'ФИО', Examen AS 'Оценка за экзамен', ID_Lesson AS 'Код предмета'

FROM Students, Progress

WHERE Students.ID_Student = Progress.ID_Student ORDER BY FIO;

Результат выполнения запроса приведен на рис. 46.

	ФИО	Оценка за экзам...	Код предмета
1	Абышкин В.В.	5	1
2	Абышкин В.В.	5	3
3	Абышкин В.В.	4	4
4	Абышкин В.В.	3	5
5	Абышкин В.В.	4	25
6	Заваровский К.Ю.	5	1
7	Заваровский К.Ю.	5	3
8	Заваровский К.Ю.	4	4
9	Иванчуков А.Г.	5	1

Рис. 46. Соединение таблиц посредством ссылочной целостности

Пример 89. Результат из примера 88 может быть получен при использовании в запросе для задания операции соединения таблиц ключевого слова ***JOIN***. Запрос с командой ***JOIN*** выглядит следующим образом:

SELECT FIO AS 'ФИО', Examen AS 'Оценка за экзамен', ID_Lesson AS 'Код предмета'

FROM Students JOIN Progress

ON Students.ID_Student = Progress.ID_Student ORDER BY FIO;

Результат выполнения запроса приведен на рис. 47.

	ФИО	Оценка за экзам...	Код предмета
1	Абышкин В.В.	5	1
2	Абышкин В.В.	5	3
3	Абышкин В.В.	4	4
4	Абышкин В.В.	3	5
5	Абышкин В.В.	4	25
6	Заваровский К.Ю.	5	1
7	Заваровский К.Ю.	5	3
8	Заваровский К.Ю.	4	4
9	Иванчуков А.Г.	5	1

Рис. 47. Соединение таблиц с помощью ключевого слова *JOIN*

Хотя выше речь шла о соединении двух таблиц, можно сформировать запросы путем соединения более чем двух таблиц.

Однако по возможности следует избегать использования соединений, так как при этом многократно увеличивается количество просматриваемых строк.

Краткие итоги

Для соединения таблиц с одноименными столбцами или таблицы с самой собой используются алиасы. Они задаются во фразе ***FROM*** через пробел после имени таблицы. При этом истинное имя таблицы в базе данных не меняется.

При внутреннем естественном соединении группируются только те строки, значения которых по соединяемым (одноименным) столбцам совпадают.

При внешнем левом соединении в результирующий набор будут выбраны все строки из левой таблицы (указываемой первой).

При совпадении значений по соединяемым (одноименным) столбцам значения второй таблицы заносятся в результирующий набор в соответствующие строки. При отсутствии совпадений в качестве значений второй таблицы проставляется значение ***NULL***.

При внешнем правом соединении в результирующий набор будут выбраны все строки из правой таблицы (указываемой второй). При совпадении значений по соединяемым (одноименным) столбцам значения первой таблицы заносятся в результирующий набор в соответствующие строки. При отсутствии совпадений в качестве значений первой таблицы проставляется значение ***NULL***.

При полном внешнем соединении в результирующий набор будут выбраны все строки, как из правой, так и из левой таблицы. При совпадении значений по соединяемым (одноименным) столбцам строка содержит значения как из левой, так и из правой таблицы. В противном случае, вместо отсутствующих значений в столбцы таблицы (левой или правой) заносится значение ***NULL***.

Часто при запросе информации необходимо осуществлять соединение таблицы с ее же копией. При соединении таблицы с ее же копией вводят алиасы таблицы.

Контрольные вопросы

1. Меняется ли имя таблицы в базе данных при использовании алиасов?
2. В чем различие между внутренним и внешним соединениями?
3. Какие различают виды соединений по способу добавления непарных строк?
4. Как выполняется операция соединения таблиц посредством ссылочной целостности?

13. Представления

Представления (***VIEW***) – это временные, производные (иначе – виртуальные) таблицы. Представления являются объектами базы данных, информация в которых не хранится постоянно, как в базовых таблицах, а формируется динамически при обращении к ним. Обычные таблицы относятся к базовым, т.е. содержащим данные и постоянно находящимся на устройстве хранения информации. Представление не может существовать само по себе, а определяется только в терминах одной или нескольких таблиц.

Представление – это предопределенный запрос, хранящийся в базе данных, который выглядит подобно обычной таблице и не требует для своего хранения дисковой памяти. Для хранения представления используется только оперативная память.

Применение представлений позволяет разработчику базы данных обеспечить каждому пользователю или группе пользователей наиболее подходящие способы работы с данными, что решает проблему простоты их использования и безопасности. Содержимое представлений выбирается из других таблиц с помощью выполнения запроса, причем при изменении значений в таблицах данные в представлении автоматически меняются. Таким образом, представление – это именованная таблица, содержимое которой является результатом запроса, заданного при описании представления. Результат выполнения этого запроса в каждый момент времени становится содержанием представления. У пользователя создается впечатление, что он работает с настоящей, реально существующей таблицей.

Создание и изменение представлений выполняются с помощью следующей команды:

<определение_ представления > ::=

{ CREATE| ALTER} VIEW имя_ представления

[(имя_столбца [,...n])]

[WITH ENCRYPTION]

AS SELECT_оператор

[WITH CHECK OPTION];

имя_столбца – по умолчанию имена столбцов в представлении соответствуют именам столбцов в исходных таблицах. Имена столбцов перечисляются через запятую, в соответствии с порядком их следования в представлении.

WITH ENCRYPTION предписывает серверу шифровать SQL-код запроса. Это гарантирует невозможность его несанкционированного просмотра и использования. Этот аргумент применяется, если при определении представления необходимо скрыть имена исходных таблиц и столбцов, а также алгоритм объединения данных.

WITH CHECK OPTION предписывает серверу исполнять проверку изменений, производимых через представление, на соответствие критериям, определенным в операторе ***SELECT***. Это означает, что не допускается выполнение изменений, которые приведут к исчезновению строки из представления. Такое случается, если для представления установлен горизонтальный фильтр и изменение данных приводит к несоответствию строки установленным фильтрам. Эта опция распространяет условие ***WHERE*** для запроса на операции обновления и вставки в описании представления. Использование аргумента ***WITH CHECK OPTION*** гарантирует, что сделанные изменения будут отображены в представлении. Если пользователь пытается выполнить изменения, приводящие к исключению строки из представления, при заданном аргументе ***WITH CHECK OPTION*** сервер выдаст сообщение об ошибке и все изменения будут отклонены.

13.1. Представления, маскирующие столбцы

Данный вид представлений ограничивает число столбцов базовой таблицы, к которым возможен доступ.

Пример 90. Обеспечить доступ пользователю к полям ***Fio, Data_Rozhd*** и ***Nomer_zachetki*** базовой таблицы ***Students***, полностью скрывая от него как содержимое, так и сам факт наличия в базовой таблице полей ***ID_Student, Adres, ID_Group, ID_Kaf.***

CREATE VIEW Stud1 AS
SELECT Fio AS ФИО,
Data_Rozhd AS [Дата рождения],
Nomer_zachetki AS [Номер зачетки]
FROM Students;

К представлению ***Stud1*** теперь можно обращаться с помощью запросов так же, как и к любой другой таблице базы данных. Например, запрос для просмотра представления ***Stud1*** имеет вид:

SELECT * FROM Stud1;

13.2. Представления, маскирующие строки

Представления могут также ограничивать доступ к строкам. Выбираемые представлением строки базовой таблицы задаются условием (предикатом) в конструкции ***WHERE*** при описании представления. Доступ через представление возможен только к строкам, удовлетворяющим условию.

Пример 91. Выбрать только те строки таблицы ***Students***, для которых значение поля ***ID_Group*** равно 14;

CREATE VIEW Stud2 AS

SELECT *

FROM Students

WHERE ID_Group = 14;

13.3. Модифицирование представлений

Данные, предъявляемые пользователю через представление, могут изменяться с помощью команд модификации DML, но при этом фактическая модификация данных будет осуществляться не в самой виртуальной таблице-представлении, а будет перенаправлена к соответствующей базовой таблице.

В общем случае следует учитывать, что обычно в представлении отображаются данные из базовой таблицы в преобразованном или усеченном виде, в результате чего применение команд модификации к таблицам-представлениям имеет некоторые особенности, рассматриваемые ниже.

Пример 92. Выбрать те строки таблицы ***Students***, где ***ID_Group*** равно 11:

CREATE VIEW STUD3 AS

SELECT *

FROM Students

WHERE ID_Group =11;

Выполним команду:

INSERT INTO Stud3 (FIO, Nomer_zachetki, ID_Group, Stipendiya)

VALUES (’Шпаликов А.А.’, ’06ВП229’, 4, 1200);

Это допустимая команда в представлении, и строка будет добавлена с помощью представления ***STUD3*** в таблицу ***Students***.

Однако когда информация будет добавлена, строка исчезнет из представления, поскольку номер группы отличен от 11. Иногда такой подход может стать проблемой, так как данные уже находятся в таблице, но пользователь их не видит и не в состоянии выполнить их удаление или модификацию.

Для исключения подобных моментов служит ***WITH CHECK OPTION*** в определении представления. Фраза размещается в определении представления, и все команды модификации будут подвергаться проверке.

Пример 93. Создать представление с проверкой команд модификации.

```
CREATE VIEW Stud4 AS
 SELECT *
 FROM Student
 WHERE N_gr = '08ВП1';
WITH CHECK OPTION;
```

Для такого представления вставка значений, приведенная выше, будет отклонена системой.

Таким образом, представление может изменяться командами модификации, но фактически модификация воздействует не на само представление, а на базовую таблицу.

Не все представления в Transact SQL могут быть модифицированы. Представление является обновляемым (модифицируемым), если в представлении могут выполняться команды модификации. В противном случае представление предназначено только для чтения при запросе.

Критерии обновляемого представления:

– основывается только на одной базовой таблице;

– содержит первичный ключ этой таблицы;

– не содержит ***DISTINCT*** в своем определении;

– не использует ***GROUP BY*** или ***HAVING*** в своем определении;

– не применяет в своем определении подзапросы;

– не использует константы или выражения среди выбранных полей вывода;

– в представление должен быть включен каждый столбец таблицы, имеющий атрибут ***NOT NULL***;

– оператор ***SELECT*** представления не использует агрегирующие (итоговые) функции, соединения таблиц, хранимые процедуры и функции, определенные пользователем;

– основывается на одиночном запросе, поэтому объединение ***UNION*** не разрешено.

Если представление удовлетворяет этим условиям, к нему могут применяться команды ***INSERT, UPDATE, DELETE***.

Модифицируемые и немодифицируемые представления создаются для различных целей.

С модифицируемыми представлениями в основном работают так же, как и с базовыми таблицами. Пользователи могут даже не знать, является ли объект, который они запрашивают, базовой таблицей или представлением. Таким образом, представление – это прежде всего средство для скрытия частей таблицы, не относящихся к потребностям данного пользователя.

Представления в режиме «только для чтения» позволяют получать и форматировать данные более рационально. Они создают целый набор сложных запросов, которые можно выполнить и повторить снова, сохраняя полученную информацию. Результаты этих запросов могут затем использоваться в других запросах, что позволит избежать сложных предикатов и снизить вероятность ошибочных действий.

Эти представления могут также иметь значение при решении задач защиты и безопасности данных. Например, можно предоставить некоторым пользователям возможность получения агрегатных данных (таких, как усредненное значение оценки студента), не показывая конкретных значений оценок и, тем более, не позволяя их модифицировать.

Пример 94. Создать обновляемое представление:

CREATE VIEW stud5

AS SELECT *

FROM Students

WHERE Stipendiya >1200;

Пример 95. Создать немодифицируемое представление с вычисляемым выражением ***«Stipendiya*2***

CREATE VIEW stud6

AS SELECT ID_Student, FIO, Nomer_zachetki, ID_Group, Stipendiya*2 AS dd

FROM Students

WHERE Stipendiya >1200;

13.4. Агрегированные представления

Создание представлений с использованием агрегирующих функций и предложения ***GROUP BY*** является удобным инструментом для непрерывной обработки и интерпретации извлекаемой информации.

Пример 96. Найти количество студентов, сдающих экзамены, количество сданных экзаменов, количество сданных предметов, средний балл по каждому предмету.

CREATE VIEW ITOGI AS

SELECT COUNT(DISTINCT ID_Lesson) AS Количество_экзаменов,

COUNT(ID_Student) AS Количество_студентов,

COUNT(Examen) AS количество_оценок,

AVG(Examen) AS средний_балл, SUM(Examen) AS суммарный_балл

FROM Progress;

Теперь требуемую информацию можно увидеть с помощью простого запроса к представлению:

SELECT * FROM ITOGI;

Результат выполнения запроса приведен на рис. 48.

	Количество_экзаменов	Количество_студентов	количество_оценок	средний_балл	суммарный_балл
1	8	19	18	4	81

Рис. 48. Представление с использованием агрегирующих функций

13.5. Представления, основанные на нескольких таблицах

Представления часто используются для объединения нескольких таблиц (базовых и/или других представлений) в одну большую виртуальную таблицу. Такое решение имеет ряд преимуществ:

– представление, объединяющее несколько таблиц, может использоваться при формировании сложных отчетов как промежуточный макет, скрывающий детали объединения большого количества исходных таблиц;

– предварительно объединенные поисковые и базовые таблицы обеспечивают наилучшие условия для транзакций, позволяют использовать компактные схемы кодов, устраняя необходимость написания для каждого отчета длинных объединяющих процедур;

– предварительно объединенные и проверенные представления уменьшают вероятность ошибок, связанных с неполным выполнением условий объединения.

Пример 97. Вывести фамилии, названия сданных предметов и оценки для каждого студента:

CREATE VIEW ocenki AS

SELECT l.Nazvanie AS Название_предмета,

s.FIO AS Фамилия_студента, p.Examen AS "оценка за экзамен"

FROM students s, Progress p, Lessons l

WHERE s.ID_Student = p.ID_Student

AND p.ID_Lesson = l.ID_Lesson;

Результат выполнения запроса приведен на рис. 49.

	Название_предмета	Фамилия_студен...	оценка за экзам...
1	Объектно-ориентированное программирование	Макарь В.А.	3
2	Объектно-ориентированное программирование	Заваровский К.Ю.	5
3	Операционные системы	Заваровский К.Ю.	5
4	Организация ЭВМ	Заваровский К.Ю.	4
5	Объектно-ориентированное программирование	Иванчуков А.Г.	5
6	Операционные системы	Иванчуков А.Г.	5
7	Организация ЭВМ	Иванчуков А.Г.	4

Рис. 49. Представление, основанное на нескольких таблицах

Теперь все названия предметов, сданных студентом, или фамилии студентов, сдававших какой-либо предмет, можно выбрать с помощью простого запроса. Например, чтобы увидеть все предметы, сданные студентом Иванчуковым А. Г, создается запрос:

SELECT Название_предмета, "оценка за экзамен"

FROM ocenki

WHERE Фамилия_студента = ’Иванчуков А.Г.’;

Результат выполнения запроса приведен на рис. 50.

	Название_предмета	оценка за экзам..
1	Объектно-ориентированное программирование	5
2	Операционные системы	5
3	Организация ЭВМ	4
4	Теория вероятности	NULL
5	Человеко-машинное взаимодействие	5
6	Базы данных	5
7	Математическая логика и теория алгоритмов	4

Рис. 50. Выборка данных из представления *Ocenki*

Краткие итоги

Представление – это предопределенный запрос, хранящийся в базе данных, который выглядит подобно обычной таблице и не требует для своего хранения дисковой памяти. Для хранения представления используется только оперативная память.

Представление является обновляемым (модифицируемым), если в представлении могут выполняться команды модификации. Иначе представление предназначено только для чтения при запросе.

Критерии обновляемого представления:

– основывается только на одной базовой таблице;

– содержит первичный ключ этой таблицы;

– не содержит ***DISTINCT***, ***GROUP BY*** или ***HAVING*** в своем определении;

– не применяет в своем определении подзапросы;

– не использует константы или выражения среди выбранных полей вывода;

– в представление должен быть включен каждый столбец таблицы, имеющий атрибут ***NOT NULL***;

– оператор ***SELECT*** представления не использует агрегирующие (итоговые) функции, соединения таблиц, хранимые процедуры и функции, определенные пользователем;

– основывается на одиночном запросе, поэтому объединение ***UNION*** не разрешено.

Если представление удовлетворяет этим условиям, к нему могут применяться операторы ***INSERT, UPDATE, DELETE***.

Модифицируемые и немодифицируемые представления создаются для различных целей.

Модифицируемые представления служат для скрытия частей таблицы, не относящихся к потребностям данного пользователя.

Немодифицируемые представления создают целый набор сложных запросов, результаты которых могут использоваться в других запросах. Это позволит избежать сложных предикатов и снизить вероятность ошибочных действий.

Представления могут также иметь значение при решении задач защиты и безопасности данных.

Представления, основанные на нескольких таблицах, имеют ряд преимуществ:

– могут использоваться при формировании сложных отчетов как промежуточный макет, скрывающий детали объединения большого количества исходных таблиц;

– обеспечивают наилучшие условия для транзакций, позволяют использовать компактные схемы кодов, устраняя необходимость написания для каждого отчета длинных объединяющих процедур;

– предварительно объединенные и проверенные уменьшают вероятность ошибок, связанных с неполным выполнением условий объединения.

Контрольные вопросы

1. Что такое представления?
2. В чем различие между представлениями и таблицами?
3. Каким целям служат представления?
4. Должно ли представление иметь одинаковое имя с таблицей, от которой порождено?
5. Возможно ли создание представления, включающего информацию из нескольких таблиц одновременно?
6. В чем различие между обновляемыми и необновляемыми представлениями?

14. Процедурная логика

Порядок выполнения запросов и хранимых процедур может изменяться с помощью операторов, приведенных в табл. 18.

Таблица 18

Операторы процедурной логики

<table>
<tr><th>Оператор</th><th>Описание</th></tr>
<tr><td>1</td><td>2</td></tr>
<tr><td>BEGIN...END</td><td>Определяет блок.
BEGIN
{оператор_SQL | блок _операторов}
END</td></tr>
<tr><td>GOTO label</td><td>Безусловный переход к метке label. Метки описываются следующим образом:
:label
GOTO label</td></tr>
<tr><td>IF...ELSE</td><td>Условный оператор.
IF логическое_выражение
{оператор_SQL | блок _операторов}
[ELSE [логическое_выражение]
{оператор_SQL | блок _операторов}</td></tr>
<tr><td>RETURN</td><td>Безусловный выход.
RETURN ([integer_expression])
Возвращаемое значение – это код возврата, причем часть значений зарезервирована под специальные состояния
<table>
<tr><th>Код</th><th>Значение</th></tr>
<tr><td>0</td><td>Все нормально</td></tr>
<tr><td>–1</td><td>Объект не найден</td></tr>
<tr><td>–2</td><td>Ошибка типа данных</td></tr>
<tr><td>–3</td><td>Процесс стал жертвой «мертвой блокировки»</td></tr>
<tr><td>–4</td><td>Ошибка доступа</td></tr>
<tr><td>–5</td><td>Синтаксическая ошибка</td></tr>
<tr><td>–6</td><td>«Некоторая» ошибка</td></tr>
<tr><td>–7</td><td>Ошибка с ресурсами (нет места)</td></tr>
<tr><td>–8</td><td>Произошла исправимая внутренняя ошибка</td></tr>
<tr><td>–9</td><td>Системный лимит исчерпался</td></tr>
<tr><td>–10</td><td>Неисправимое нарушение внутренней целостности</td></tr>
<tr><td>–11</td><td>То же</td></tr>
<tr><td>–12</td><td>Разрушение таблицы или индекса</td></tr>
<tr><td>–13</td><td>Разрушение базы данных</td></tr>
<tr><td>–14</td><td>Ошибка оборудования</td></tr>
</table></td></tr>
</table>

Окончание табл. 18

1	2
WHILE	Цикл с предусловием. ***WHILE логическое_выражение*** ***{оператор_SQL | блок _операторов}*** ***[BREAK]*** ***{оператор_SQL | блок _операторов}*** [***CONTINUE***]
...BREAK	Выход из цикла ***WHILE***
...CONTINUE	Продолжение цикла ***WHILE***
DECLARE	Позволяет объявлять локальные переменные. Этот оператор может стоять не только в начале процедуры, но и где угодно в ее теле
PRINT	Выдает заданное значение на экран. ***IF EXISTS (SELECT ID_Kaf*** ***FROM Students*** ***WHERE ID_Kaf = 1)*** ***PRINT 'Кафедра МОиПЭВМ'*** Длина строки с сообщением не должна превышать 255 символов
CASE	Позволяет выражениям принимать значение в зависимости от условий. ***CASE*** является стандартной возможностью ANSI SQL-92 Простое ***CASE***-выражение: ***CASE expression*** ***WHEN expression1 THEN exression1*** ***[[WHEN expression2 THEN expression2[..]]*** ***[ELSE expressionN]*** ***END***

Пример 98. По введенному краткому названию кафедры вывести ее полное название:

SELECT [Название кафедры] =
CASE NameKaf
WHEN 'МОиПЭВМ' THEN 'Математическое обеспечение и применение ЭВМ'
WHEN 'САПР' THEN 'Системы автоматизированного проектирования'
WHEN 'ИнОУп' THEN 'Информационное обеспечение управления'
WHEN 'ВТ' THEN 'Вычислительная техника'
ELSE 'Нет такой кафедры'
END;

15. Хранимые процедуры

Хранимая процедура (Stored procedure) – программа, которая выполняется внутри базы данных и может предпринимать сложные действия на основе информации, задаваемой пользователем. Поскольку хранимые процедуры выполняются непосредственно на сервере базы данных, обеспечивается более высокое быстродействие, чем при выполнении тех же операций средствами клиента базы данных.

Хранимая процедура объединяет запросы и процедурную логику (операторы присваивания, логического ветвления и т.п.) и хранится в базе данных.

Одна процедура может быть использована в любом количестве клиентских приложений, что позволяет существенно сэкономить трудозатраты на создание прикладного программного обеспечения и эффективно применять стратегию повторного использования кода. Так же, как и любые процедуры в стандартных языках программирования, хранимые процедуры могут иметь входные и выходные параметры или не иметь их.

Преимущества выполнения в базе данных хранимых процедур вместо отдельных команд *Transact SQL*:

– необходимые команды уже содержатся в базе данных;

– все они прошли этап синтаксического анализа и находятся в исполняемом формате;

– хранимые процедуры поддерживают модульное программирование, так как позволяют разбивать большие задачи на самостоятельные, более мелкие и удобные в управлении части;

– хранимые процедуры могут вызывать другие хранимые процедуры и функции;

– хранимые процедуры могут быть вызваны из прикладных программ других типов;

– как правило, хранимые процедуры выполняются быстрее, чем последовательность отдельных команд;

– хранимые процедуры проще использовать: они могут состоять из десятков и сотен команд, но для их запуска достаточно указать всего лишь имя нужной хранимой процедуры. Это позволяет уменьшить размер запроса, посылаемого от клиента на сервер, а значит, и нагрузку на сеть.

Хранимые процедуры вызываются клиентской программой, другой хранимой процедурой или триггером. Разработчик может управлять правами доступа к хранимой процедуре, разрешая или запрещая ее выполнение. Изменять код хранимой процедуры разрешается только ее владельцу или члену фиксированной роли базы данных. При необходимости можно передать права владения ею от одного пользователя к другому.

15.1. Типы хранимых процедур

В SQL Server 2005 имеется несколько типов хранимых процедур:

– ***Системные хранимые процедуры*** предназначены для выполнения различных административных действий. Практически все действия по администрированию сервера выполняются с их помощью. Системные хранимые процедуры имеют префикс sp_, хранятся в системной базе данных и могут быть вызваны в контексте любой другой базы данных.

– ***Пользовательские хранимые процедуры*** реализуют какие-либо действия. Хранимые процедуры – полноценный объект базы данных. Поэтому каждая хранимая процедура располагается в конкретной базе данных, где и выполняется.

– ***Временные хранимые процедуры*** существуют лишь некоторое время, после чего автоматически уничтожаются сервером. Они делятся на ***локальные*** и ***глобальные***.

– ***Локальные временные хранимые процедуры*** могут быть вызваны только из того соединения, в котором созданы. При создании такой процедуры ей необходимо дать имя, начинающееся с одного символа #. Как и все временные объекты, хранимые процедуры этого типа автоматически удаляются при отключении пользователя, перезапуске или остановке сервера.

– ***Глобальные временные хранимые процедуры*** доступны для любых соединений сервера, на котором имеется такая же процедура. Для ее определения достаточно дать ей имя, начинающееся с символов ##. Удаляются эти процедуры при перезапуске или остановке сервера, а также при закрытии соединения, в контексте которого они были созданы.

15.2. Создание, изменение и удаление хранимых процедур

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

<определение_процедуры>::=
{CREATE | ALTER } PROC[EDURE] имя_процедуры
[;номер]
[{@имя_параметра тип_данных } [VARYING]
[=значение_по_умолчанию][OUTPUT]][,...n]
[WITH { RECOMPILE | ENCRYPTION | RECOMPILE,
ENCRYPTION }]
[FOR REPLICATION]
AS
Тело процедуры;

Номер в имени – это идентификационный номер хранимой процедуры, однозначно определяющий ее в группе процедур. Для удобства управления процедурами логически однотипные хранимые процедуры можно группировать, присваивая им одинаковые имена, но разные идентификационные номера.

Для передачи входных и выходных данных в создаваемой хранимой процедуре могут использоваться параметры, имена которых, как и имена локальных переменных, должны начинаться с символа **@**. В одной хранимой процедуре можно задать множество параметров, разделенных запятыми. В теле процедуры не должны применяться локальные переменные, чьи имена совпадают с именами параметров этой процедуры.

Для определения типа данных, который будет иметь соответствующий параметр хранимой процедуры, годятся любые типы данных SQL, включая определенные пользователем.

OUTPUT означает, что соответствующий параметр предназначен для возвращения данных из хранимой процедуры. Указание ключевого слова ***OUTPUT*** предписывает серверу при выходе из хранимой процедуры присвоить текущее значение параметра локальной переменной, которая была указана при вызове процедуры в качестве значения параметра. При указании ключевого слова ***OUTPUT*** значение соответствующего параметра при вызове про-

цедуры может быть задано только с помощью локальной переменной. Не разрешается использование любых выражений или констант, допустимое для обычных параметров.

Необязательное ключевое слово ***VARYING*** определяет заданное значение по умолчанию для определенного ранее параметра.

Ключевое слово ***DEFAULT*** представляет собой значение, которое будет принимать соответствующий параметр по умолчанию. Таким образом, при вызове процедуры можно не указывать явно значение соответствующего параметра.

Так как сервер кэширует план исполнения запроса и компилированный код, при последующем вызове процедуры будут использоваться уже готовые значения. Однако в некоторых случаях все же требуется выполнять перекомпиляцию кода процедуры. Указание ключевого слова ***RECOMPILE*** предписывает системе создавать план выполнения хранимой процедуры при каждом ее вызове.

Параметр ***FOR REPLICATION*** востребован при репликации данных и включении создаваемой хранимой процедуры в качестве статьи в публикацию.

Ключевое слово ***ENCRYPTION*** предписывает серверу выполнить шифрование кода хранимой процедуры, что может обеспечить защиту от использования авторских алгоритмов, реализующих работу хранимой процедуры.

Ключевое слово ***AS*** размещается в начале собственно тела хранимой процедуры. В теле процедуры могут применяться практически все команды SQL, объявляться транзакции, устанавливаться блокировки и вызываться другие хранимые процедуры. Выход из хранимой процедуры можно осуществить посредством команды ***RETURN***.

Удаление хранимой процедуры осуществляется командой:

DROP PROCEDURE {имя_процедуры} [,...n];

15.2.1. Выполнение хранимой процедуры

Для выполнения хранимой процедуры используется команда:

EXEC [UTE] имя_процедуры [;номер]
[[@имя_параметра=]{значение | @имя_переменной}
[OUTPUT]|[DEFAULT]][,...n]

Если вызов хранимой процедуры не является единственной командой в пакете, то присутствие команды ***EXECUTE*** обязательно. Более того, эта команда требуется для вызова процедуры из тела другой процедуры или триггера.

Использование ключевого слова ***OUTPUT*** при вызове процедуры разрешается только для параметров, которые были объявлены при создании процедуры с ключевым словом ***OUTPUT***.

Если при вызове процедуры для параметра указывается ключевое слово ***DEFAULT***, то будет использовано значение по умолчанию. Естественно, указанное слово ***DEFAULT*** разрешается только для тех параметров, для которых определено значение по умолчанию.

Из синтаксиса команды ***EXECUTE*** видно, что имена параметров могут быть опущены при вызове процедуры. Однако в этом случае пользователь должен указывать значения для параметров в том же порядке, в каком они перечислялись при создании процедуры. Присвоить параметру значение по умолчанию, просто пропустив его при перечислении, нельзя. Если же требуется опустить параметры, для которых определено значение по умолчанию, достаточно явного указания имен параметров при вызове хранимой процедуры. Более того, таким способом можно перечислять параметры и их значения в произвольном порядке.

При вызове процедуры указываются либо имена параметров со значениями, либо только значения без имени параметра. Их комбинирование не допускается.

Пример 99. Процедура без параметров. Разработать процедуру для получения названий экзаменов и оценок, полученных студентом Иванчуковым А. Г.:

CREATE Procedure ExamResults

AS

SELECT s.FIO AS ’ФИО’, l.Nazvanie AS ’Дисциплина’, p.Examen AS ’Оценка за экзамен’

FROM Students AS s INNER JOIN

Progress AS p ON p.ID_Student = s.ID_Student INNER JOIN

Lessons AS l ON l.ID_Lesson = p.ID_Lesson

WHERE s.FIO = ’Иванчуков А.Г.’;

Для обращения к процедуре можно использовать команду

EXECUTE ExamResults;

Результат выполнения запроса приведен на рис. 51.

	ФИО	Дисциплина	Оценка за экзам...
1	Иванчуков А.Г.	Объектно-ориентированное программирование	5
2	Иванчуков А.Г.	Операционные системы	5
3	Иванчуков А.Г.	Организация ЭВМ	4
4	Иванчуков А.Г.	Теория вероятности	NULL
5	Иванчуков А.Г.	Человеко-машинное взаимодействие	5
6	Иванчуков А.Г.	Базы данных	5
7	Иванчуков А.Г.	Математическая логика и теория алгоритмов	4

Рис. 51. Процедура без параметров

Пример 100. Процедура без параметров. Создать процедуру для уменьшения размера стипендии на 10 %:

CREATE Procedure Reduce

AS

UPDATE Students SET Stipendiya = Stipendiya * 0.9

WHERE Stipendiya IS NOT NULL;

Для обращения к процедуре можно использовать команду:

EXECUTE Reduce;

Пример 101. Процедура с входным параметром. Разработать процедуру для получения названий экзаменов и оценок, полученных заданным студентом:

CREATE Procedure ExamResult

@FIO varchar(70)

AS

SELECT s.FIO AS ’ФИО’, l.Nazvanie AS ’Дисциплина’, p.Examen AS ’Оценка за экзамен’

FROM Students AS s INNER JOIN

Progress AS p ON p.ID_Student = s.ID_Student INNER JOIN

Lessons AS l ON l.ID_Lesson = p.ID_Lesson

WHERE s.FIO = @FIO;

Для обращения к процедуре можно использовать команды:

EXECUTE ExamResult ’Токунов А.Г.’ или

ExamResult @FIO =’Токунов А.Г.’;

Результат выполнения запроса приведен на рис. 52.

	ФИО	Дисциплина	Оценка за экзам...
1	Токунов А.Г.	Объектно-ориентированное программирование	5
2	Токунов А.Г.	Операционные системы	5
3	Токунов А.Г.	Организация ЭВМ	5

Рис. 52. Процедура с входным параметром

Пример 102. Процедура с входными параметрами. Создать процедуру для выдачи списка студентов, получивших определенную оценку по определенному экзамену:

CREATE Procedure Subject

@Subject varchar(50), @Mark tinyint

AS

SELECT s.FIO AS 'ФИО', l.Nazvanie AS 'Дисциплина', p.Examen AS 'Оценка за экзамен'

FROM Students AS s

INNER JOIN

Progress AS p ON p.ID_Student = s.ID_Student

INNER JOIN

Lessons AS l ON l.ID_Lesson = p.ID_Lesson

WHERE l.Nazvanie = @Subject AND p.Examen = @Mark;

Для обращения к процедуре можно использовать команду:

EXEC Subject 'Объектно-ориентированное программирование', 5;

Результат выполнения запроса приведен на рис. 53.

	ФИО	Дисциплина	Оценка за экзам...
1	Заваровский К.Ю.	Объектно-ориентированное программирование	5
2	Иванчуков А.Г.	Объектно-ориентированное программирование	5
3	Абышкин В.В.	Объектно-ориентированное программирование	5
4	Токунов А.Г.	Объектно-ориентированное программирование	5

Рис. 53. Процедура с входными параметрами

Пример 103. Процедура с входными параметрами и значениями по умолчанию. Создать процедуру для выдачи списка студентов, получивших определенную оценку по определенному экзамену. По умолчанию вывести фамилии студентов, получивших оценку «3» по дисциплине «Алгебра и геометрия»:

CREATE Procedure ExamResultsDef

@Subject varchar(50)= VARYING 'Алгебра и геометрия',

@Mark tinyint = 3

AS

SELECT s.FIO AS 'ФИО', l.Nazvanie AS 'Дисциплина', p.Examen AS 'Оценка за экзамен'

FROM Students AS s INNER JOIN

Progress AS p ON p.ID_Student = s.ID_Student INNER JOIN

Lessons AS l ON l.ID_Lesson = p.ID_Lesson

WHERE (@Subject IS NOT NULL AND l.Nazvanie = @Subject AND p.Examen = @Mark) OR

(@Subject IS NULL AND p.Examen = @Mark);

Для обращения к процедуре можно использовать команды:

1. ***EXEC ExamResultsDef*** – в этом случае выводятся значения по умолчанию, т.е. заданные в процедуре, – оценка «3» и дисциплина «Алгебра и геометрия».

2. ***EXEC ExamResultsDef @Subject = 'Объектно-ориентированное программирование', @Mark =5*** – в этом случае выводится список студентов, получивших оценку «5» по дисциплине «Объектно-ориентированное программирование».

3. ***EXEC ExamResultsDef @Subject = 'Объектно-ориентированное программирование' –*** выводится список студентов, получивших оценку «3» по дисциплине «Объектно-ориентированное программирование».

4. ***EXEC ExamResultsDef @Mark = 5*** выводится список студентов, получивших оценку «5» по дисциплине «Алгебра и геометрия».

Пример 104. Процедура с входными и выходными параметрами. Вывести число студентов у определенного куратора.

CREATE Procedure StudentsNum

@Num smallint OUTPUT,

@CuratorSn varchar(20),

@CuratorN varchar(20),

@CuratorP varchar(20)

AS

SELECT @Num = COUNT(*)

FROM Students s

INNER JOIN Groups g ON g.ID_Group = s.ID_Group

INNER JOIN Teachers t ON t.ID_Teacher = g.Kurator

WHERE t.Familia = @CuratorSn AND t.Imja = @CuratorN AND t.Surname = @CuratorP

Для обращения к процедуре можно использовать команды:

DECLARE @Result smallint

EXECUTE StudentsNum @Result OUTPUT, 'Самуйлов, 'Сергей', 'Владимирович'

PRINT CAST (@Result AS varchar (40));

В результате выполнения процедуры выводится сообщение `Число студентов 1.`

Пример 105. Использование вложенных процедур. Создать процедуру для определения куратора группы, в которой учится определенный студент.

Сначала разработаем процедуру для определения групп и их кураторов:

CREATE PROCEDURE Curator

@Grp VARCHAR(10),

@Srn VARCHAR(20) OUTPUT

AS

SELECT @Srn = Familia

FROM Teachers

INNER JOIN Groups ON Groups.Kurator = Teachers.ID_Teacher

WHERE Groups.Name_group = @Grp;

Затем создадим процедуру, определяющую студентов и их кураторов:

CREATE PROCEDURE StudentsCurator

@FIO VARCHAR(70),

@Crtr VARCHAR(20) OUTPUT

AS

DECLARE @GNm VARCHAR(10)

SELECT @GNm = Name_group

FROM Students s

INNER JOIN Groups g ON g.ID_Group = s.ID_Group
WHERE s.FIO = @FIO
EXEC Curator @GNm,@Crtr OUTPUT

Вызов процедуры осуществляется следующим образом:

DECLARE @Crtr VARCHAR(20)
EXECUTE StudentsCurator 'Додонов А.С.', @Crtr OUTPUT
PRINT @Crtr;

15.3. Получение информации о процедурах

Несколько системных процедур выдают информацию из системных таблиц о сохраненных процедурах.

1. Процедура *sp_help.*

С помощью системной процедуры ***sp_help*** можно получить отчет о сохраненной процедуре. Например, пользователь может получить информацию о сохраненной процедуре ***proc1*** из созданной базы данных с помощью следующей команды:

sp_help proc1;

2. Процедура *sp_helptext.*

Чтобы увидеть текст (тело) сохраненной процедуры, нужно вызвать системную процедуру ***sp_helptext***:

sp_helptext proc1;

Чтобы увидеть текст системной процедуры, нужно вызвать процедуру ***sp_helptext*** из базы данных ***sybsystemprocs*.**

Краткие итоги

Хранимая процедура (Stored procedure) – программа, которая выполняется внутри базы данных и может предпринимать сложные действия на основе информации, задаваемой пользователем. Хранимые процедуры выполняются непосредственно на сервере базы данных. За счет этого обеспечивается более высокое быстродействие, чем при выполнении тех же операций средствами клиента базы данных.

Хранимая процедура объединяет запросы и процедурную логику (операторы присваивания, логического ветвления и т.п.) и хранится в базе данных.

Преимущества выполнения в базе данных хранимых процедур вместо отдельных операторов SQL:

– необходимые операторы уже содержатся в базе данных, они прошли этап синтаксического анализа и находятся в исполняемом формате;

– хранимые процедуры позволяют разбивать большие задачи на самостоятельные, более мелкие и удобные в управлении части;

– хранимые процедуры могут вызывать другие хранимые процедуры и функции;

– хранимые процедуры могут быть вызваны из прикладных программ других типов;

– хранимые процедуры выполняются быстрее, чем последовательность отдельных операторов;

– хранимые процедуры проще использовать: они могут состоять из десятков и сотен команд, но для их запуска достаточно указать всего лишь имя нужной хранимой процедуры. Это позволяет уменьшить размер запроса, посылаемого от клиента на сервер, а значит, и нагрузку на сеть.

Типы хранимых процедур:

– Системные хранимые процедуры.

– Пользовательские хранимые процедуры.

– Временные хранимые процедуры. Они делятся на локальные и глобальные.

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

CREATE | ALTER PROC[EDURE] имя_процедуры;

Удаление хранимой процедуры осуществляется командой

DROP PROCEDURE {имя_процедуры.

Для выполнения хранимой процедуры используется команда

EXEC [UTE] имя_процедуры.

Контрольные вопросы

1. Что такое хранимая процедура?
2. Где выполняются хранимые процедуры?
3. Как активизируются хранимые процедуры?
4. В чем преимущества использования хранимых процедур?
5. Какие типы хранимых процедур имеются в SQL Server 2005?

16. Управление транзакциями

Современные СУБД являются многопользовательскими. Следовательно, всегда есть возможность одновременного обращения нескольких пользователей к одной базе данных и даже к одним и тем же данным. При этом возникает масса проблем, связанных с попытками одновременного изменения или удаления данных.

Во избежание таких ситуаций в СУБД вводится понятие **транзакции**.

Транзакция – это неделимая, с точки зрения воздействия на базу данных, последовательность операций обработки данных, которая выполняется как единое целое и переводит базу данных из одного целостного состояния в другое.

Разработчик приложений исходя из смысла обработки данных определяет, какая последовательность операций составляет единое целое, т.е. транзакцию.

16.1. Определение транзакций

SQL Server поддерживает три вида определения транзакций:

– явное;

– автоматическое;

– подразумеваемое.

По умолчанию SQL Server работает в режиме автоматического начала транзакций, где каждая команда рассматривается как отдельная транзакция. Если команда выполнена успешно, то ее изменения фиксируются. Если при выполнении команды произошла ошибка, то сделанные изменения отменяются и система возвращается в первоначальное состояние.

Когда пользователю понадобится создать транзакцию, включающую несколько команд, он должен явно указать транзакцию.

Сервер работает только в одном из двух режимов определения транзакций: автоматическом или подразумевающемся. Он не может находиться в режиме исключительно явного определения транзакций. Этот режим работает поверх двух других.

Для установки режима автоматического определения транзакций используется команда

SET IMPLICIT_TRANSACTIONS OFF;

При работе в режиме неявного (подразумевающегося) начала транзакций SQL Server автоматически начинает новую транзакцию, как только будет завершена предыдущая. Установка режима подразумевающегося определения транзакций выполняется посредством команды

SET IMPLICIT_TRANSACTIONS ON;

16.2. Явные транзакции

Явные транзакции требуют, чтобы пользователь указал начало и конец транзакции, используя следующие команды:

1) ***начало транзакции****:* в журнале транзакций фиксируются первоначальные значения изменяемых данных и момент начал транзакции:

BEGIN TRAN[SACTION] [имя_транзакции]

2) ***конец транзакции COMMIT {[TRAN[SACTION][имя_транзакции]|.[WORK]};***

3) ***создание внутри транзакции точки сохранения*****:** СУБД сохраняет состояние БД в текущей точке и присваивает сохраненному состоянию имя точки сохранения:

SAVE TRAN[SACTION] имя_точки_сохранения;

4) прерывание транзакции; когда сервер встречает эту команду, происходит откат транзакции, восстанавливается первоначальное состояние системы и в журнале транзакций отмечается, что транзакция была отменена:

ROLLBACK [TRAN[SACTION]

[имя_транзакции | имя_точки_сохранения];

Эта команда отменяет все изменения, в базу данных после оператора ***BEGIN TRANSACTION*** или отменяет изменения, внесенные в базу данных после точки сохранения. Транзакция возвращается к месту, где был выполнен оператор ***SAVE TRANSACTION.***

Пример 114. Использование точек сохранения

BEGIN TRAN

SAVE TRANSACTION point1

В точке ***point1*** сохраняется первоначальное состояние таблицы ***Students.***

Добавим в таблицу ***Students*** новую запись. В точке ***point2*** сохраняется новое состояние таблицы ***Students***:

INSERT INTO Students (FIO, Nomer_zachetki, ID_Group, ID_Kaf)
VALUES ('Носиков А.В.', '06ВП219', 9, 1)

SAVE TRANSACTION point2

SELECT * FROM Students;

В таблице ***Students*** появилась новая запись.

Выполним откат транзакции в точку сохранения ***point1***.

ROLLBACK TRANSACTION point1

SELECT * FROM Students;

Оператор ***SELECT*** покажет таблицу ***Students*** без студента Носикова А. В., т.е. происходит возврат в первоначальное состояние таблицы:

COMMIT;

Первоначальное состояние фиксируется.

Краткие итоги

Современные СУБД являются многопользовательскими. Во избежание ситуаций, когда несколько пользователей одновременно обращаются к одной базе данных и даже к одним и тем же данным, в СУБД вводится понятие **транзакции**.

Транзакция – это неделимая, с точки зрения воздействия на базу данных, последовательность операций обработки данных, которая выполняется как единое целое и переводит базу данных из одного целостного состояния в другое.

Разработчик приложений исходя из смысла обработки данных определяет, какая последовательность операций составляет единое целое, т.е. транзакцию.

SQL Server поддерживает три вида определения транзакций:

– явное;

– автоматическое;

– подразумеваемое.

По умолчанию SQL Server работает в режиме автоматического начала транзакций, где каждая команда рассматривается как отдельная транзакция. Если команда выполнена успешно, то ее изменения фиксируются. Если при выполнении команды произошла

ошибка, то сделанные изменения отменяются и система возвращается в первоначальное состояние.

Когда пользователю понадобится создать транзакцию, включающую несколько команд, он должен явно указать транзакцию.

Сервер работает только в одном из двух режимов определения транзакций: автоматическом или подразумевающемся. Он не может находиться в режиме исключительно явного определения транзакций. Этот режим работает поверх двух других.

Явные транзакции требуют, чтобы пользователь указал начало и конец транзакции, используя следующие команды:

1) ***начало транзакции – BEGIN TRAN[SACTION [имя_транзакции]***

2) ***конец транзакции – COMMIT{[TRAN[SACTION] [имя_транзакции] | [WORK]}***

3) ***создание внутри транзакции точки сохранения –***

SAVE TRAN[SACTION] имя_точки_сохранения

4) ***прерывание транзакции –***

ROLLBACK [TRAN[SACTION] [имя_транзакции | имя_точки_сохранения];

Контрольные вопросы

1. Что такое транзакция?
2. Какие виды определения транзакций поддерживает SQL Server?
3. Какие команды используются для явного задания транзакций?

17. Триггеры

Триггеры – это предварительно определенное действие или последовательность действий, ***автоматически*** осуществляемых при выполнении операций обновления, добавления или удаления данных.

Исключительно важно в этом определении слово «***автоматически***». Ни пользователь, ни приложение не могут активизировать триггер, он выполняется автоматически, когда пользователь или приложение выполняют с базой данных определенные действия.

Триггер – это специальный вид хранимой процедуры. Триггеры обеспечивают проверку любых изменений на корректность, прежде чем эти изменения будут приняты.

Каждый триггер привязывается к конкретной таблице. Все производимые им модификации данных рассматриваются как одна транзакция. В случае обнаружения ошибки или нарушения целостности данных происходит откат этой транзакции. Тем самым внесение изменений запрещается. Отменяются также все изменения, уже сделанные триггером.

Создать триггер может только владелец базы данных. Это ограничение позволяет избежать случайного изменения структуры таблиц, способов связи с ними других объектов и т.п.

17.1. Компоненты триггера

1. ***Ограничения***, для реализации которых создается триггер.

2. ***Событие***, которое будет характеризовать возникновение ситуации, требующей проверки ограничений. Триггерные события чаще всего связаны с изменением состояния базы данных и состоят из вставки, удаления и обновления строк в таблице. События могут учитываться и дополнительные условия (например, добавление записи только с отрицательным значением).

3. ***Предусмотренное действие*** осуществляется за счет выполнения процедуры или последовательности процедур, с помощью которых реализуется логика**,** требуемая для реализации ограничений.

Триггер выполняется неявно в каждом случае возникновения триггерного события. Приведение его в действие называют ***запуском триггера***. С помощью триггеров достигаются следующие цели:

– проверка корректности введенных данных и выполнение сложных ограничений целостности данных, которые трудно, если вообще возможно, поддерживать с помощью ограничений целостности, установленных для таблицы;

– выдача предупреждений, напоминающих о необходимости выполнения некоторых действий при обновлении таблицы, реализованном определенным образом.

17.2. Типы триггеров

Существует три типа триггеров:

1. **Insert** – определяет действия, которые будут выполняться после добавления новой записи в таблицу.

2. **Update** – определяет действия, которые будут выполняться после изменения записи таблицы.

3. **Delete** – определяет действия, которые будут выполняться после удаления записи из таблиц.

Часто в СУБД определяется большее число событий, с которыми можно связать триггеры. Например, до вставки, после вставки, до изменения, после изменения и т.д.

17.3. Создание триггеров

Основной формат команды ***CREATE TRIGGER***:

<Определение_триггера>::=

CREATE TRIGGER [имя_триггера]

ON имя_таблицы

{ FOR | AFTER | INSTEAD OF} {[INSERT] [,] [UPDATE] [,] [DELETE]}

[WITH ENCRYPTION]

AS SQL_операторы

Или используя предложение IF UPDATE

<Определение_триггера>::=

CREATE TRIGGER [имя_триггера]

ON имя_таблицы

{ FOR | AFTER | INSTEAD OF} {[INSERT] [,] [UPDATE] [,] [DELETE]}

[WITH ENCRYPTION]
AS
IF UPDATE (имя_столбца)
[{AND | OR} UPDATE (имя_столбца)...]
SQL_операторы;

CREATE TRIGGER [имя_триггера] – создается новый триггер с именем ***имя_триггера***

ON имя_таблицы – объявляется таблица или представление, от которых зависит триггер.

WITH ENCRYPTION имеет тот же смысл, что и для хранимых процедур, он скрывает исходный текст тела триггера.

{ FOR | AFTER | INSTEAD OF} – указывает, когда должен запускаться триггер. Ключевые слова ***FOR*** и ***AFTER*** являются синонимами. Предложение ***AFTER*** показывает, что триггер запускается только после успешного выполнения операции по модификации данных (и других каскадно запускаемых действий и проверок ограничений). Триггер ***INSTEAD OF*** может полностью заменить операцию по модификации данных. При этом триггер запускается вместо операции по модификации, которая запустила триггер. Триггер ***INSTEAD OF DELETE*** нельзя использовать, если удаление вызывает каскадные действия. Доступ к столбцам ***TEXT*** или ***IMAGE*** имеют только триггеры ***INSTEAD OF*** .

IF UPDATE (имя_столбца) [{AND | OR} UPDATE (имя_столбца)...] – позволяет выбрать конкретный столбец, запускающий триггер. Триггеры, специфичные для столбца, запускаются только при операциях ***INSERT*** или ***UPDATE,*** но не ***DELETE.***

Конструкции ***FOR {INSERT, UPDATE, DELETE}*** определяют, на какую команду будет реагировать триггер. При его создании должна быть указана хотя бы одна команда. Допускается создание триггера, реагирующего на две или на все три команды.

Неправильно написанные триггеры могут привести к серьезным проблемам, таким, например, как появление «мертвых» блокировок. Триггеры способны длительное время блокировать множество ресурсов, поэтому следует обратить особое внимание на сведение к минимуму конфликтов доступа.

В большинстве СУБД действуют следующие ограничения:

– Нельзя использовать в теле триггера операции создания объектов базы данных (новой базы данных, новой таблицы, новой хранимой процедуры, нового триггера, новых представлений).

– Нельзя использовать в триггере команду удаления объектов ***DROP*** для всех типов базовых объектов базы данных.

– Нельзя использовать в теле триггера команды изменения базовых объектов ***ALTER TABLE, ALTER DATABASE.***

– Нельзя изменять права доступа к объектам базы данных, т.е. выполнять команду ***GRAND*** или ***REVOKE***.

– Нельзя создать триггер для представления (***VIEW***) .

– В отличие от хранимых процедур триггер не может возвращать никаких значений, он запускается автоматически сервером и не может связаться самостоятельно ни с одним клиентом.

– Внутри триггера не допускается выполнение восстановления резервной копии БД или журнала транзакций.

Выполнение этих команд не разрешено, так как они не могут быть отменены в случае отката транзакции, в которой выполняется триггер.

Преимущества использования триггеров:

1. Триггеры всегда выполняются при совершении соответствующих действий. Разработчик продумывает использование триггеров при проектировании базы данных и может больше не вспоминать о них при разработке приложения для доступа к данным. Если для работы с этой же базой данных нужно создать новое приложение, триггеры и там будут отрабатывать заданные ограничения.

2. При необходимости триггеры можно изменять централизованно непосредственно в базе данных. Пользовательские программы, использующие данные из этой базы данных, не потребуют модернизации.

3. Система обработки данных, использующая триггеры, обладает лучшей переносимостью в архитектуру клиент-сервер за счет меньшего объема требуемых модификаций.

17.4. Программирование триггеров

При выполнении команд добавления, изменения и удаления записей сервер создает две специальные таблицы: ***inserted*** и ***deleted***. В них содержатся списки строк, которые будут вставлены или удалены по завершении транзакции. 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 ее удаление. В таблице ***inserted*** не окажется ни одной строки;

– команда ***UPDATE*** – при ее 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 помнить, 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 определенная 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 фиксации изменений, внесенных при выполнении транзакции, следует использовать команду ***COMMIT TRANSACTION***.

Для удаления триггера используется команда

DROP TRIGGER {имя_триггера} [,...n].

17.5. Примеры использования триггеров

Пример 106. Реализовать ограничение на значение.

При добавлении записи в таблицу ***Students*** автоматически проверяется количество студентов в заданной группе и, если их количество больше 20, то происходит откат транзакции. Если же студентов в данной группе меньше 20, то происходит увеличение количества студентов на 1 и происходит добавление записи в таблицу ***Students***.

Команда вставки записи в таблицу ***Students*** может быть такой:

INSERT INTO Students (FIO, Nomer_zachetki, ID_Group, Stipendiya) VALUES ('Ильин С.В.', '08ВП131', 2, 1250);

```
CREATE TRIGGER InsertStudent
 ON Students FOR Insert
 AS
 DECLARE @ID INT
 IF @@ROWCOUNT=1
 BEGIN
 SELECT @ID=ID_Group
 FROM INSERTED
 BEGIN
        IF 20>(SELECT Kol_stud
               FROM Groups
               WHERE ID_Group=@ID
               )
```

```
        BEGIN
        UPDATE Groups
        SET Kol_stud=Kol_stud+1
        WHERE ID_Group=@ID
        PRINT 'студент успешно добавлен в данную группу'
    END
    ELSE
    BEGIN
    ROLLBACK TRANSACTION
    PRINT 'Группа переполнена!Выберите другую группу!'
    END
    END
    END;
```

Пример 107. Создать триггер для обработки операции удаления записи из таблицы ***Students***, например, такой команды:

DELETE FROM Students WHERE ID_Student=82;

При удалении студента из группы количество студентов в группе уменьшается на единицу:

```
CREATE TRIGGER TriggerDelete
 ON Students FOR Delete
 AS
 DECLARE @ID INT, @ID_Grup INT
 IF @@ROWCOUNT=1
 BEGIN
      SELECT @ID=ID_Group
      FROM DELETED
 UPDATE Groups
 SET Kol_stud=Kol_stud-1
 WHERE ID_Group=@ID
 PRINT 'студент успешно удален из группы'
 END;
```

17.6. Использование хранимых процедур в триггерах

Хранимые процедуры могут быть активизированы не только пользовательскими приложениями, но и триггерами.

Пример 108. Создадим процедуру, обновляющую количество студентов в группе:

```
CREATE PROCEDURE UpdateKolStud
@group INT
AS
DECLARE @newKolStud SMALLINT
BEGIN
SELECT @newKolStud = COUNT(*) FROM Students WHERE ID_Group = @group
UPDATE Groups SET Kol_Stud = @newKolStud WHERE ID_Group = @group
END;
```

Теперь создадим триггер, который будет срабатывать при удалении студента из базы данных или добавлении студента в базу данных:

```
CREATE TRIGGER KolStudTrigger
ON Students
AFTER INSERT, DELETE
AS
DECLARE @gr1 INT, @gr2 INT
if @@rowcount = 1
BEGIN
    SELECT @Gr1 = ID_Group FROM deleted
    SELECT @Gr2 = ID_Group FROM inserted
    IF (SELECT DISTINCT ID_Group FROM deleted) IS NOT NULL
        EXEC UpdateKolStud @group = @gr1;
    IF (SELECT DISTINCT ID_Group FROM inserted) IS NOT NULL
        EXEC UpdateKolStud @group = @gr2;
END;
```

Вид таблицы до транзакции приведен на рис. 54.

ID_Student	FIO	Data_Rozhd	Adres	Nomer_zachetki	ID_Group	ID_Kaf	Stipendiya
7	Макарь В.А.	NULL	NULL	06ВП118	4	1	583
10	Ивкин И.Ю.	NULL	NULL	06ВП110	4	1	1239
11	Заваровский К....	NULL	NULL	06ВП109	4	1	583
12	Заваровский К.В.	NULL	NULL	06ВП108	4	1	874

Рис. 54. Таблица *Students*

В группе № 4 обучается 24 человека.

Результат выполнения запроса приведен на рис. 55.

ID_Group	Name_group	Kol_stud	Kurator
1	06ВП2	25	1
2	06ВП1	23	2
3	06ВВ1	22	5
4	07ВП1	24	6
5	07ВП2	24	3

Рис. 55. Таблица *Groups*

Удалим студента № 7, который учится в группе № 4:

DELETE FROM Students WHERE ID_Student = 7;

После удаления записи о студенте количество студентов в группе уменьшилось на единицу.

Вид таблицы после выполнения транзакции приведен на рис. 56.

ID_Group	Name_group	Kol_stud	Kurator
1	06ВП2	25	1
2	06ВП1	23	2
3	06ВВ1	22	5
4	07ВП1	23	6
5	07ВП2	24	3

Рис. 56. Таблица *Groups*

Теперь вновь добавим удаленного нами студента:

INSERT INTO Students (FIO, Nomer_zachetki, ID_Group, Stipendiya) VALUES ('Макарь В.А.', '06ВП118', 4, 1200);

Результат выполнения запроса приведен на рис. 57.

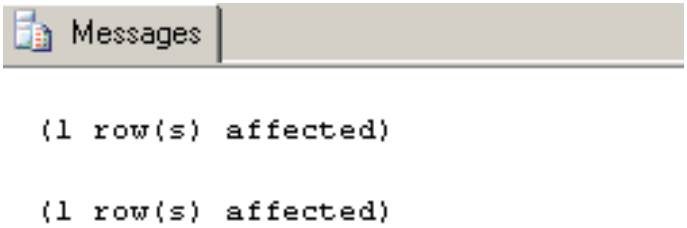


Рис. 57. Добавление строки в таблицу

Вид таблицы после транзакции приведен на рис. 58.

ID_Group	Name_group	Kol_stud	Kurator
1	06ВП2	25	1
2	06ВП1	23	2
3	06ВВ1	22	5
4	07ВП1	24	6
5	07ВП2	24	3

Рис. 58. Таблица *Groups*

Краткие итоги

Триггеры – это предварительно определенное действие или последовательность действий, ***автоматически*** осуществляемых при выполнении операций обновления, добавления или удаления данных.

Триггер – это специальный вид хранимой процедуры. Триггеры обеспечивают проверку любых изменений на корректность, прежде чем эти изменения будут приняты.

Каждый триггер привязывается к конкретной таблице. Все производимые им модификации данных рассматриваются как одна транзакция. В случае обнаружения ошибки или нарушения целостности данных происходит откат этой транзакции. Тем самым внесение изменений запрещается. Отменяются также все изменения, уже сделанные триггером.

Создать триггер может только владелец базы данных.

Компоненты триггера:

1. Ограничения.
2. Событие.
3. Предусмотренное действие.

Типы триггеров – **Insert**, **Update**, **Delete;**

Триггеры создаются с помощью команды ***CREATE TRIGGER***:

CREATE TRIGGER [имя_триггера]

ON имя_таблицы

{ FOR | AFTER | INSTEAD OF} {[INSERT] [,] [UPDATE] [,] [DELETE]}

[WITH ENCRYPTION]

AS SQL_операторы;

Неправильно написанные триггеры могут привести к серьезным проблемам, таким, например, как появление «мертвых» блокировок. Триггеры способны длительное время блокировать множество ресурсов, поэтому следует обратить особое внимание на сведение к минимуму конфликтов доступа.

В большинстве СУБД действуют следующие ограничения:

– в теле триггера нельзя использовать операции создания, удаления и изменения объектов базы данных;

– нельзя изменять права доступа к объектам базы данных, т.е. выполнять команду ***GRAND*** или ***REVOKE***;

– нельзя создать триггер для представления (***VIEW***);

– триггер не может возвращать никаких значений, он запускается автоматически сервером и не может связаться самостоятельно ни с одним клиентом;

– внутри триггера не допускается выполнение восстановления резервной копии БД или журнала транзакций.

Выполнение этих команд не разрешено, так как они не могут быть отменены в случае отката транзакции, в которой выполняется триггер.

В триггерах могут использоваться хранимые процедуры.

Контрольные вопросы

1. Что такое триггер?
2. Каковы компоненты триггера?
3. Триггеры каких типов существуют?
4. Привести пример использования триггера при работе с БД.
5. Допускается ли внутри триггера применение команд управления транзакциями?
6. В чем преимущества использования триггеров?

18. Управление пользователями базы данных

Стабильная система управления пользователями – обязательное условие безопасности данных, хранящихся в любой реляционной СУБД.

После проектирования логической структуры базы данных, связей между таблицами, ограничений целостности и других структур необходимо определить круг пользователей, которые будут иметь доступ к базе данных.

В системе SQL Server 2005 организована двухуровневая настройка ограничения доступа к данным.

На первом уровне необходимо создать учетную запись пользователя (***login***). Это позволяет пользователю подключиться к серверу, но не дает автоматического доступа к базам данных.

На втором уровне для каждой базы данных SQL Server 2005 на основании учетной записи необходимо создать запись пользователя. На основе прав, выданных пользователю как пользователю базы данных (***user***), его регистрационное имя (***login***) получает доступ к соответствующей базе данных. С помощью учетной записи пользователя осуществляется подключение к SQL Server, после чего определяются его уровни доступа для каждой базы данных в отдельности.

На уровне сервера система безопасности оперирует следующими понятиями:

– аутентификация*;*

– учетная запись*;*

– встроенные роли сервера.

На уровне базы данных применяются следующие понятия;

– пользователь базы данных;

– фиксированная роль базы данных;

– пользовательская роль базы данных.

SQL Server предлагает два режима аутентификации пользователей:

1. Режим аутентификации средствами Windows – Windows Authentication.

2. Режим аутентификации средствами SQL Server – SQL Server Authentication.

18.1. Создание пользователя

Для создания пользователя в среде MS SQL Server необходимо выполнить следующие действия:

1. Создать в базе данных учетную запись пользователя, указав для него пароль и принятое по умолчанию имя базы данных (процедура ***sp_addlogin***).

2. Добавить этого пользователя во все необходимые базы данных (процедура ***sp_adduser***).

3. Предоставить ему в каждой базе данных соответствующие привилегии (команда ***GRANT***) .

18.2. Создание новой учетной записи

Создание новой учетной записи может быть произведено с помощью системной хранимой процедуры

sp_addlogin

[@login=] 'учетная_запись'

[, [@password=] 'пароль']

[, [@defdb=] 'база_данных_по_умолчанию'];

После завершения аутентификации и получения идентификатора учетной записи (***login***) пользователь считается зарегистрированным, и ему предоставляется доступ к серверу.

Пример 109. Создание пользователя ***student***

sp_addlogin 'student', 'stud', Institute;

18.3. Добавление пользователя в базу данных

Для каждой базы данных, к объектам которой пользователь намерен получить доступ, учетная запись пользователя (***login***) ассоциируется с пользователем (***user***) конкретной базы данных. Это осуществляется посредством процедуры

sp_adduser

[@loginame=] 'учетная_запись'

[, [@name_in_db=] 'имя_пользователя']

[, [@grpname=] 'имя_роли'];

Пример 110. Добавить пользователя ***student*** к объектам базы данных ***Institute***:

USE Institute;

sp_adduser 'student';

Пользователь, который создает объект в базе данных (таблицу, хранимую процедуру, представление), становится его владельцем. Владелец объекта ***(database object owner – dbo***) имеет все права доступа к созданному им объекту. Чтобы пользователь мог создать объект, владелец базы данных (***dbo***) должен предоставить ему соответствующие права. Полное имя создаваемого объекта включает в себя имя создавшего его пользователя.

Владелец объекта не имеет специального пароля или особых прав доступа. Он неявно имеет полный доступ, но должен явно предоставить доступ другим пользователям.

SQL Server позволяет передавать права владения от одного пользователя другому с помощью процедуры:

sp_changeobjectowner

[@objname=] 'имя_объекта'

[@newowner=] 'имя_владельца';

18.4. Роли

В системе SQL Server существуют дополнительные объекты – роли, которые определяют уровень доступа к объектам SQL Server. Роль позволяет объединить в одну группу пользователей, выполняющих одинаковые функции.

В SQL Server реализовано два вида стандартных ролей:

1. На уровне сервера.
2. На уровне баз данных.

При установке SQL Server создаются фиксированные роли сервера (например, ***sysadmin*** с правом выполнения любых функций SQL Server) и фиксированные роли базы данных (например, ***db_owner*** с правом полного доступа к базе данных). Среди фиксированных ролей базы данных существует роль ***public***, которая имеет специальное назначение, поскольку ее членами являются все пользователи, имеющие доступ к базе данных.

Можно включить любую учетную запись SQL Server 2005 (***login***) или учетную запись Windows NT в любую роль сервера.

Роли базы данных позволяют объединять пользователей в одну административную единицу и работать с ней как с обычным пользователем. Можно назначить права доступа к объектам базы данных для конкретной роли, при этом автоматически все члены этой роли наделяются одинаковыми правами.

В роль базы данных можно включить пользователей SQL Server, роли SQL Server, пользователей Windows NT.

Различные действия по отношению к роли осуществляются с помощью специальных процедур:

- создание новой роли:

sp_addrole
[@rolename=] 'имя_роли'
[, [@ownername=] 'имя_владельца']

- добавление пользователя к роли:

sp_addrolemember
[@rolename=] 'имя_роли',
[@membername=] 'имя_пользователя'

- удаление пользователя из роли:

sp_droprolemember
[@rolename=] 'имя_роли',
[@membername=] 'имя_пользователя'

- удаление роли:

sp_droprole
[@rolename=] 'имя_роли';

18.5. Категории прав пользователей

При подключении к SQL Server все возможные действия пользователей определяются правами (привилегиями, разрешениями), выданными их учетной записью, группе или роли, в которых они состоят.

Привилегиями, или правами, называются действия, которые пользователь имеет право выполнять в отношении данной таблицы базы данных или представления.

Права можно разделить на три категории:

– права на доступ к объектам;

– права на выполнение команд;

– неявные права.

Работа с данными и выполнение хранимых процедур требуют наличия класса доступа, называемого ***правами на доступ к объектам* баз *данных***. Под объектами подразумеваются таблицы, столбцы таблиц, представления, хранимые процедуры.

Для управления полномочиями пользователя на доступ к объектам базы данных используется команда:

<предоставление_привилегий>::=

GRANT { [объектные_привилегии] [,…] |

[системные_привилегии] }

[ON { [объект] [(Столбец [,…])]}]

TO { имя_получателя [,…] | роль [,...] | PUBLIC}

[WITH GRANT OPTION]

[AS {имя_группы | имя_роли }];

объектные_привилегии – или права на доступ к объектам баз данных. Предоставляются привилегии для выполнения различных операций в любых сочетаниях (за исключением ***ALL [PRIVILEGES]).***

ALL [PRIVILEGES] – предоставляются все привилегии, назначенные в настоящий момент указанным пользователям и/или для указанных объектов базы данных.Использование данного предложения может привести к нечеткости программирования. Предложение ***ALL*** могут использовать только пользователи с системными ролями ***SYSADMIN*** и ***DB_OWNER*** или владелец объекта.

{SELECT | DELETE | INSERT | UPDATE} – указанному пользователю присваивается указанная привилегия доступа к указанному объекту (например, к таблице или представлению). При указании привилегий уровня столбца список столбцов следует заключить в скобки.

Для различных объектов применяются разные наборы прав доступа к ним:

- ***SELECT, INSERT, UPDATE, DELETE, REFERENCES*** – для таблицы или представления;
- ***SELECT, UPDATE*** – для конкретного столбца таблицы или представления;
- ***EXECUTE*** – для хранимых процедур и функций.

Право ***INSERT*** позволяет вставлять новые строки в таблицу или представление. Выдается только на уровне таблицы или представления и не может быть выдано на уровне столбца.

Право ***UPDATE*** выдается либо на уровне таблицы (что позволяет изменять в ней все данные), либо на уровне отдельного столбца (это разрешает изменять данные только в пределах столбца).

Право ***DELETE*** позволяет удалять строки из таблицы или представления. Выдается только на уровне таблицы или представления, но не может быть выдано на уровне столбца.

Право ***SELECT*** разрешает выборку данных и может выдаваться как на уровне таблицы, так и на уровне отдельного столбца.

Право ***REFERENCES*** предоставляет возможность ссылаться на указанный объект. Применительно к таблицам разрешает создавать внешние ключи, ссылающиеся на первичный ключ или уникальный столбец этой таблицы.

EXECUTE – предоставляется привилегия выполнять хранимую процедуру или пользовательскую функцию. Естественно, кроме этого владелец хранимой процедуры может просматривать и изменять ее код.

Системные_привилегии – или права на выполнение команд SQL. Этот класс прав контролирует возможность создания базы данных, объектов в базе данных и выполнения процедуры резервного копирования. Предоставляются привилегии на выполнение следующих команд:

CREATE DATABASE

CREATE TABLE

CREATE VIEW

CREATE DEFAULT

CREATE RULE

CREATE PROCEDURE

BACKUP DATABASE

BACKUP LOG;

Привилегия на выполнение инструкции ***CREATE*** также подразумевает привилегию на выполнение соответствующих команд ***ALTER*** и ***DROP.***

***ON { [объект] [(Столбец [,…])]}* –** указывается объект, к которому назначается право доступа. Это предложение не требуется при назначении системных привилегий.

TO { имя_получателя [,…] | роль [,...] | PUBLIC} – имя пользователя или роли, которым назначается привилегия. Через запятую можно указать несколько получателей привилегии.

PUBLIC – подразумевается, что это роль всех пользователей.

WITH GRANT OPTION – поможет пользователю, которому предоставляются права, назначить права на доступ к объекту другим пользователям. Его использование требует особой осторожности, поскольку при этом владелец теряет контроль над предоставлением прав на доступ другим пользователям. Лучше всего ограничить круг пользователей, обладающих возможностью управлять назначением прав. Используется только применительно к объектным привилегиям.

***AS {имя_группы | имя_роли }* –** позволяет указать участие пользователя в роли, обеспечивающей предоставление прав другим пользователям.

Привилегии можно присваивать только в текущей базе данных. Нельзя присваивать привилегии одновременно в нескольких базах данных.

18.6. Неявные права

Выполнение некоторых действий не требует явного разрешения и доступно по умолчанию. Эти действия могут быть выполнены только членами ролей сервера или владельцами объектов в базе данных.

Неявные права не предоставляются пользователю непосредственно, он получает их лишь при определенных обстоятельствах. Например, пользователь может стать владельцем объекта базы данных, только если сам создаст объект либо если кто-то другой передаст ему право владения своим объектом. Таким образом, владелец объекта автоматически получит права на выполнение любых действий с объектом, в том числе и на предоставление доступа к объекту другим пользователям. Эти права нигде не указываются, выполнять любые действия позволяет только факт владения объектом.

18.7. Отмена предоставленных пользователям привилегий

В языке SQL для отмены привилегий, предоставленных пользователям посредством оператора ***GRANT***, используется оператор ***REVOKE***. С помощью этого оператора могут быть отменены все или некоторые из привилегий, полученных указанным пользователем раньше. Оператор ***REVOKE*** имеет следующий формат:

<отмена_привилегий>::=

REVOKE [GRANT OPTION FOR]

{<привилегия>[,...n]

| ALL PRIVILEGES}

ON имя_объекта

FROM {<идентификатор_пользователя> [,...n]| PUBLIC}

[CASCADE]

[AS {имя_группы | имя_роли }];

GRANT OPTION FOR – позволяет для всех привилегий, переданных в исходном операторе ***GRANT*** фразой ***WITH GRANT OPTION***, отменять возможность их передачи независимо от самих привилегий;

ALL PRIVILEGES – означает, что для указанного пользователя отменяются все привилегии, предоставленные ему ранее тем пользователем, который ввел данный оператор;

ON имя_объекта – отменяется право доступа пользователя к указанному объекту;

FROM {<идентификатор_пользователя> [,...n]| PUBLIC} – указываются пользователи или роли, теряющие указанную привилегию.

CASCADE – удаляются все привилегии, которые иначе могли бы остаться у других пользователей. Поскольку наличие привилегии необходимо для создания определенных объектов, вместе с ее удалением можно лишиться права, за счет использования которого был образован тот или иной объект (подобные объекты называются «брошенными»). Если в результате выполнения оператора ***REVOKE*** могут появиться брошенные объекты (например представления), оно будет отменено при условии, что в нем не указывается ключевое слово ***CASCADE***. Если ключевое слово ***CASCADE*** в операторе

присутствует, то для любых брошенных объектов, возникающих при выполнении исходного оператора ***REVOKE***, будут автоматически выданы операторы ***DROP***.

[AS {имя_группы | имя_роли }] – указываются права, в соответствии с которыми отменяются привилегии.

18.8. Запрещение доступа

При необходимости пользователю можно запретить доступ к данным или командам. Тогда аннулируются все разрешения на доступ, полученные им.

Для запрещения доступа к объектам базы данных используется команда ***DENY:***

DENY {ALL [PRIVILEGES]| | <привилегия>
[,...n]}
{ [(имя_столбца [,...n])]
ON { имя_таблицы |
имя_просмотра}
| ON {имя_таблицы | имя_просмотра }
| ON {имя_хранимой_процедуры |
имя_внешней_процедуры}}
TO {имя_пользователя | имя_группы |
имя_роли}
[,...n]
[CASCADE]

Для запрещения выполнения команд SQL применяется оператор:

<запрещение_выполнения>::=
DENY {ALL | <команда>[,...n]}
TO {имя_пользователя | имя_группы |
имя_роли} [,...n];

Синтаксис инструкции ***DENY*** аналогичен синтаксису инструкции ***REVOKE.*** Однако они отличаются тем, что ***REVOKE*** отменяет предоставленные привилегии, а ***DENY*** явно запрещает доступ.

Пример 111. Определить для пользователя, созданного в примере 110, привилегии выполнения операций выборки данных и добавления данных в таблицу ***Students:***

GRANT SELECT, INSERT ON Students TO student;

Пример 112. Создать нового пользователя ***prepodavatel.***

SP_ADDLOGIN ’prepodavatel’,’123654’,’Institute’;

USE ’Institute’;

SP_ADDUSER ’prepodavatel’;

Определить для него привилегии выполнения всех операций для таблиц ***Teachers*** и ***Progress***.

GRANT ALL ON Teachers TO prepodavatel WITH GRANT OPTION;

GRANT ALL ON Progress TO prepodavatel;

Пример 113. Создать представление, в котором вычисляется средняя оценка для каждого студента, сдававшего экзамены.

CREATE VIEW Sr_Mark (ФИО, средняя_оценка) as

SELECT FIO, AVG(Examen)

FROM Students, Progress

WHERE Students.Id_Student=Progress.Id_Student;

Предоставить пользователю ***prepodavatel*** привилегию выборки данных из этого представления (т.е. привилегию выполнения команды ***SELECT:***

GRANT SELECT ON Sr_Mark TO prepodavatel;

Краткие итоги

После проектирования логической структуры базы данных, связей между таблицами, ограничений целостности и других структур необходимо определить круг пользователей, которые будут иметь доступ к базе данных.

В системе SQL Server организована двухуровневая настройка ограничения доступа к данным.

На первом уровне создается учетная запись пользователя (***login***). На втором уровне для каждой базы данных SQL Server на основании учетной записи необходимо создать запись пользователя. На основе прав, выданных пользователю как пользователю базы данных (***user***), его регистрационное имя (***login***) получает доступ к

соответствующей базе данных. С помощью учетной записи пользователя осуществляется подключение к SQL Server.

Для создания пользователя необходимо:

1. Создать в базе данных учетную запись пользователя с помощью процедуры ***sp_addlogin***).

2. Добавить этого пользователя во все необходимые базы данных (процедура ***sp_adduser***).

3. Предоставить ему в каждой базе данных соответствующие привилегии (команда ***GRANT***) .

Роль позволяет объединить в одну группу пользователей, выполняющих одинаковые функции.

В SQL Server реализовано два вида стандартных ролей – на уровне сервера и на уровне баз данных.

Различные действия по отношению к роли осуществляются с помощью специальных процедур:

– создание новой роли – ***sp_addrole;***

– добавление пользователя к роли – ***sp_addrolemember;***

– удаление пользователя из роли – ***sp_droprolemember;***

– удаление роли – ***sp_droprole***

Привилегии, или ***права,*** – это действия, которые пользователь имеет право выполнять в отношении данной таблицы базы данных или представления.

Существуют права трех категорий:

– права на доступ к объектам;

– права на выполнение команд;

– неявные права.

Работа с данными и выполнение хранимых процедур требуют наличия класса доступа, называемого ***правами на доступ к объектам* баз *данных***. Под объектами подразумеваются таблицы, столбцы таблиц, представления, хранимые процедуры.

Для управления полномочиями пользователя на доступ к объектам базы данных используется команда ***GRANT***.

В языке SQL для отмены привилегий, предоставленных пользователям посредством оператора ***GRANT***, используется оператор ***REVOKE***. С помощью этого оператора могут быть отменены все

или некоторые из привилегий, полученных указанным пользователем раньше.

Для запрещения доступа к объектам базы данных и для запрещения выполнения команд SQL используется команда ***DENY***.

Контрольные вопросы

1. Как добавить нового пользователя в базу данных?
2. Кто является владельцем объектов базы данных?
3. Какие категории прав (привилегий) существуют в SQL Server?
4. Каким образом подтверждаются и отменяются привилегии пользователей на доступ к конкретной таблице БД?
5. Какими привилегиями обладает пользователь по отношению к созданной по его требованию таблице?
6. Каким образом осуществляется разделение привилегий на доступ к отдельным столбцам таблицы?

Список литературы

1. Microsoft SQL Server 2005. Реализация и обслуживание : учебный курс Microsoft / пер. с англ. – М. : Русская редакция ; СПб. : Питер, 2007.

2. Грофф, Дж. SQL: Полное руководство / Дж. Грофф, П. Вайнберг ; пер. с англ. – 2-е изд., перераб. и доп.. – Киев : Издательская группа BHV, 2005.

3. Форта, Бен. Освой самостоятельно SQL. 10 минут на урок, 3-е изд. / Бен. Форта ; пер. с англ. – М. : Издательский дом «Вильямс», 2005.

4. Хендерсон, К. Профессиональное руководство по SQL Server: хранимые процедуры, XML, HTML / К. Хендерсон. – СПб. : Питер, 2005.

5. Хендерсон, К. Профессиональное руководство по SQL Server: структура и реализация / К. Хендерсон ; пер. с англ. – М. : Издательский дом «Вильямс», 2006.

Предметный указатель

СОДЕРЖАНИЕ

Учебное издание

Казакова Ирина Анатольевна

Основы языка Transact SQL

Редактор *Т. В. Веденеева*
Корректор *Ж. А. Лубенцова*
Компьютерная верстка *Р. Б. Бердниковой*

Подписано в печать 09.07.10. Формат 60x84 1/16.
Усл. печ. л. 9,53. Тираж 500.
Заказ № 454.

Издательство ПГУ.
440026, Пенза, Красная, 40.