МЕЖГОСУДАРСТВЕННОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ОБРАЗОВАНИЯ
«БЕЛОРУССКО-РОССИЙСКИЙ УНИВЕРСИТЕТ»

Кафедра «Программное обеспечение информационных технологий»

БАЗЫ ДАННЫХ

Методические рекомендации к лабораторным работам для студентов направлений подготовки 09.03.01 «Информатика и вычислительная техника» и 09.03.04 «Программная инженерия» очной формы обучения

Часть 2

Могилев 2022

УДК 004.65
ББК 32.973.26-0.18.2
Б17

Рекомендовано к изданию
учебно-методическим отделом
Белорусско-Российского университета

Одобрено кафедрой «Программное обеспечение информационных технологий» «28» декабря 2021 г., протокол № 6

Составители: канд. техн. наук, доц. К. В. Захарченков;
канд. техн. наук, доц. Т. В. Мрочек

Рецензент канд. техн. наук, доц. В. М. Ковальчук

Методические рекомендации содержат описание двенадцати лабораторных работ», выполняемых во втором семестре изучения дисциплины «Базы данных». Рассматриваются основы работы с Microsoft SQL Server и языком Transact-SQL.

Учебно-методическое издание

БАЗЫ ДАННЫХ

Часть 2

Ответственный за выпуск	В. В. Кутузов
Корректор	И. В. Голубцова
Компьютерная верстка	Н. П. Полевничая

Подписано в печать . Формат 60×84/16. Бумага офсетная. Гарнитура Таймс.
Печать трафаретная. Усл. печ. л. . Уч.-изд. л. . Тираж 21 экз. Заказ №

Издатель и полиграфическое исполнение:
Межгосударственное образовательное учреждение высшего образования
«Белорусско-Российский университет».
Свидетельство о государственной регистрации издателя,
изготовителя, распространителя печатных изданий
№ 1/156 от 07.03.2019.
Пр-т Мира, 43, 212022, г. Могилев.

Содержание

10 Технология создания баз данных на основе промышленной СУБД MS SQL Server

Цель: получить навыки установки Microsoft SQL Server; научиться создавать базу данных на основе скрипта SQL.

Теоретические положения

Microsoft SQL Server – система управления реляционными базами данных (СУБД), использующая язык структурированных запросов Transact-SQL (T-SQL) [1–11]. Установить Microsoft SQL Server Developer Edition можно с официального сайта (https://docs.microsoft.com/ru-ru/sql/tools/).

Для подключения к серверу баз данных и выполнения большинства действий над базой данных (БД) используется среда Microsoft SQL Server Management Studio (SSMS).

Генерация SQL-скрипта для создания схемы базы данных на основе модели в CASE-средстве. В Sparx Enterprise Architect необходимо выбрать пункт меню «Package» → «Database Engineering → Generate Package DDL …». Откроется диалоговое окно «Generate DDL», в котором можно установить ряд параметров. Опции генерации кода можно просмотреть на вкладке «Options». Выбирается способ записи в один файл «Single File» и указывается к нему путь. После этого следует нажать кнопку «Generate» для автоматической генерации SQL-скрипта. Полученный файл можно просмотреть с помощью программы «Блокнот» и использовать в СУБД MS SQL Server для автоматического создания схемы БД.

БД в SQL Server состоит из двух частей:

1) файл данных – файл, имеющий расширение .mdf, в котором находятся все основные объекты БД (таблицы, индексы, представления и т. д.);

2) файл журнала транзакций – файл, имеющий расширение .ldf, который содержит журнал, где фиксируются все действия с БД (записываются сведения о процессе работы с транзакциями (контроль целостности данных, состояния базы данных до и после выполнения транзакций). Данный файл предназначен для восстановления БД в случае её выхода из строя.

Создание новой базы данных в SSMS. Создать новую базу данных можно с использованием диалоговых средств SSMS [4, с. 126–140].

Для создания файла БД в обозревателе объектов SSMS следует нажать правую клавишу мыши на папке «Databases» (Базы данных) и в контекстном меню выбрать пункт «New Database» или «Создать базу данных». В появившемся окне нужно указать имя БД, определить ее владельца (по умолчанию), задать путь доступа к файлам БД .mdf и .ldf.

Далее на вкладке создания нового запроса нужно выполнить SQL-скрипт, сгенерированный в Sparx Enterprise Architect.

Один экземпляр SSMS может управлять несколькими базами данных. Вся информация о базах данных, управляемых SSMS, хранится в системной базе данных master.

Создать новую базу данных можно также с использованием оператора T-SQL CREATE DATABASE [4, с. 93–104, 116–118].

Удалить базу данных можно с помощью контекстного меню, выбрав пункт *Удалить*, или с использованием оператора T-SQL DROP DATABASE.

Отсоединение и *присоединение* БД используются для переноса БД между различными экземплярами SSMS. Разработанная под управлением сервера S1 БД может быть отсоединена от S1, скопирована на диск другого компьютера и присоединена к серверу S2 на втором компьютере. Для присоединения БД нужно в обозревателе объектов SSMS выбрать «Базы данных» → «Присоединить», для отсоединения БД – выбрать «Базы данных», выделить имя отсоединяемой БД и в контекстном меню выбрать «Задачи» → «Отсоединить».

Создать таблицу в SSMS можно на диаграмме баз данных либо с помощью обозревателя объектов. В обозревателе нужно раскрыть вкладку с созданной БД, раскрыть вкладку «Таблицы» и в появившемся после нажатия правой клавиши мыши контекстном меню выбрать «Создать таблицу». В появившемся окне определения полей новой таблицы указать следующее:

– Column Name – имя поля, начинающееся с буквы и не содержащее различных специальных символов и знаков препинания. Если имя поля содержит пробелы, то оно автоматически заключается в квадратные скобки-ограничители;

– Data Type – тип данных поля [4, 5, 11];

– Allow NULL – разрешить значения NULL. Если эта опция поля включена, то в случае незаполнения поля в него будет подставлено значение NULL.

После создания таблиц можно перейти к построению схемы БД. Для этого в обозревателе объектов нужно выбрать «Диаграммы баз данных» и в контекстном меню – «Создать диаграмму базы данных» добавить все таблицы в окно схемы БД. Для определения связей следует «ухватиться» за поле главной таблицы и «перетащить» его на поле подчиненной таблицы. При определении связи появляется окно «Create Relationship», в котором задается название связи в поле «Relationship name».

Рассмотрим рабочие окна интерфейса SQL Server Management Studio, часто используемые на практике. Откройте меню Вид.

Окно **Обозреватель объектов** – предназначено для выполнения административных операций с серверами, БД и объектами баз данных: обзор серверов, создание и размещение объектов, управление источниками данных, просмотр журналов. Поддерживается фильтрация и сортировка объектов.

Окно **Подробности обозревателя объектов** – предназначено для просмотра различной сводной информации по объектам, с которыми ведется работа, и т. д.

Окно **Обозреватель решений** – позволяет получать информацию о хранении и организации скриптов и соответствующие сведенья о соединении в проектах, называемых скриптами SQL Server. Несколько скриптов SQL Server можно хранить в виде решений и по мере развития скриптов для управления ими использовать систему управления версиями.

Окно **Браузер (Обозреватель) шаблонов** – позволяет создавать запросы на основе существующих шаблонов. Можно создавать пользовательские запросы или изменять существующие шаблоны в соответствии с текущими задачами.

Задание

Необходимо сгенерировать схему базы данных из Enterprise Architect в MS SQL Server.

Содержание отчета: тема и цель работы; схема БД в MS SQL Server.

Контрольные вопросы

1 Перечислить типы файлов БД в SQL Server и пояснить их назначение.
2 С помощью каких операторов T-SQL создается и удаляется БД?
3 Для чего применяется отсоединение и присоединение БД?
4 Сколько схем БД можно создать? Что такое ограничения целостности БД?

11 Создание таблиц средствами SQL

Цель: получить навыки создания таблиц средствами T-SQL.

Теоретические положения

Процесс создания таблицы начинается с проектирования ее будущей структуры. В процессе проектирования необходимо определить:

– для хранения каких данных предназначена создаваемая таблица;

– каким образом будет обеспечиваться целостность данных в ней. Для этого следует определить ограничения на значения столбцов (CONSTRAINTS).

Для создания таблицы используется следующая инструкция T-SQL [4, 11]:

```
CREATE TABLE
        { database_name.schema_name.table_name | schema_name.table_name
          | table_name }
        (  {  <column_definition>
              | <computed_column_definition>
              | [ <table_constraint> ] [ ,... n ]
        )
```

Определение каждого столбца таблицы, в синтаксисе обозначенное как <column_definition>, имеет следующий формат:

```
<column_definition> ::=
column_name <data_type>
   [ COLLATE collation_name ]
   [ NULL | NOT NULL ]
[
   [ CONSTRAINT constraint_name ] DEFAULT constant_expression ]
```

```
   | [ IDENTITY [ ( seed , increment ) ]
]
   [ <column_constraint> ]
```

Подобным образом необходимо описать каждый столбец в таблице. В обязательном порядке определяют имя столбца (column_name) и тип хранимых в нем данных (data_type).

Аргумент computed_column_expression обозначает выражение, определяющее значение вычисляемого столбца, который представляет собой виртуальный столбец, автоматически вычисляемый на основе выражения, использующего другие столбцы той же таблицы. Сами вычисляемые столбцы и их значения могут храниться в таблице (при наличии признака PERSISTED) или их значения могут не присутствовать в БД, а вычисляться только при отображении данных таблицы. Выражение для вычисления может быть именем невычисляемого столбца, константой, функцией, переменной или любой их комбинацией, соединенной одним или несколькими операторами. Выражение не может быть вложенным запросом или содержать псевдонимы типов данных. Результат большинства выражений допускает значение NULL, даже если используются только столбцы, для которых значение NULL запрещено, т. к. при возможном переполнении или потере точности может получаться значение NULL.

Ограничение <column_constraint> на значение столбца (ограничение на уровне столбца) должно начинаться с ключевого слова CONSTRAINT, после которого указывают имя ограничения и определяют тип ограничения:

```
<column_constraint> ::=
[ CONSTRAINT constraint_name ]
{
   { PRIMARY KEY | UNIQUE }
      [ CLUSTERED | NONCLUSTERED ]
      [  WITH FILLFACTOR = fillfactor    ]
  | [ FOREIGN KEY ]
      REFERENCES [ schema_name. ] referenced_table_name [ ( ref_column ) ]
  [ ON DELETE { NO ACTION | CASCADE | SET NULL | SET DEFAULT } ]
  [ ON UPDATE { NO ACTION | CASCADE | SET NULL | SET DEFAULT } ]
      [ NOT FOR REPLICATION ]
  | CHECK [ NOT FOR REPLICATION ] ( logical_expression )
}
```

Ограничения могут носить произвольные названия, но, как правило, применяются следующие префиксы: PK – для PRIMARY KEY, FK – для FOREIGN KEY, CK – для CHECK, UQ – для UNIQUE, DF – для DEFAULT.

Задавать имена ограничений необязательно, т. к. при установке соответствующих атрибутов SQL Server автоматически определяет их имена. Но, зная имя ограничения, можно к нему обращаться, например, для его удаления.

Раздел <table_constraint> позволяет задать ограничения на значения столбцов на уровне таблицы.

Рассмотрим аргументы инструкции для создания таблицы.

PRIMARY KEY – определяет столбец как первичный ключ таблицы. В качестве альтернативы можно определить столбец как уникальный, воспользовавшись ключевым словом UNIQUE. При необходимости можно также указать, будет ли индекс, создаваемый для данного ограничения, кластерным (ключевое слово CLUSTERED) или некластерным (NONCLUSTERED). Кластерный индекс можно определить только для одного ограничения. Для ограничений PRIMARY KEY по умолчанию создается кластерный индекс (CLUSTERED), а для ограничений UNIQUE – некластерный (NONCLUSTERED). Если создается индекс, необходимо также указать степень заполнения его страниц (ключевое слово WITH FILLFACTOR).

FOREIGN KEY REFERENCES – обеспечивает ссылочную целостность данных, определяет столбец как внешний ключ в дочерней таблице. С помощью ключевого слова REFERENCES необходимо указать имя родительской таблицы, на которую будет ссылаться создаваемая таблица. Дополнительно потребуется указать ее столбцы, которые будут связаны с данным столбцом внешнего ключа. Ограничения FOREIGN KEY могут ссылаться только на столбцы PRIMARY KEY или UNIQUE в родительской таблице или на столбцы, на которые имеются ссылки в индексе UNIQUE INDEX родительской таблицы.

DEFAULT – определяет значение по умолчанию (constant_expression), используемое, если при вводе строки явно не указано другое значение. DEFAULT может применяться к любым столбцам, кроме имеющих тип timestamp или обладающих свойством IDENTITY.

IDENTITY – осуществляет заполнение столбца идентификаторов автоматически. При этом можно также указать начальное значение (seed) и приращение (increment). Значения seed и increment по умолчанию (1,1). Этот атрибут может назначаться для столбцов числовых типов INT, SMALLINT, BIGINT, TINYINT, DECIMAL и NUMERIC. Для каждой таблицы можно создать только один столбец идентификаторов.

Предложение NOT FOR REPLICATION может указываться для свойства IDENTITY, а также ограничений FOREIGN KEY и CHECK. В случае, когда указано NOT FOR REPLICATION, этот столбец не будет автоматически заполняться для строк, вставляемых в таблицу в процессе репликации, так что эти строки сохранят свои значения.

ROWGUIDCOL – указывает, что данный столбец будет использоваться для хранения глобального идентификационного номера GUID строки. Свойство ROWGUIDCOL может быть присвоено только столбцу типа uniqueidentifier. Тип данных uniqueidentifier позволяет получать и хранить в переменной или в столбце таблицы уникальные значения. Инициализировать столбец или переменную типа uniqueidentifier можно посредством функции NEWID(). К столбцу со значениями типа данных uniqueidentifier можно обращаться, используя в запросе ключевое слово ROWGUIDCOL, чтобы указать, что столбец содержит значения идентификаторов (то ключевое слово не генерирует никаких значений).

Таблица может содержать несколько столбцов типа uniqueidentifier, но только один из них может иметь ключевое слово ROWGUIDCOL. При указании ROWGUIDCOL не выполняется автоматическое формирование значений для новых строк, вставляемых в таблицу.

COLLATE collation_name – задает параметры сортировки для столбца. Могут использоваться параметры сортировки Windows или параметры сортировки SQL. Аргумент collation_name применим только к столбцам типа char, varchar, text, nchar, nvarchar и ntext. Если этот аргумент не указан, столбцу назначаются либо параметры сортировки определяемого пользователем типа, если столбец принадлежит к определяемому пользователем типу данных, либо установленные по умолчанию параметры сортировки для базы данных.

Ограничение UNIQUE позволяет определить в столбце наличие только уникальных значений. В таблице может быть несколько ограничений UNIQUE.

CHECK – ограничение, обеспечивающее целостность домена путем ограничения диапазона возможных значений, которые могут быть введены в столбец.

Пример создания таблицы с ограничениями на значения столбцов:

```
CREATE TABLE Production.OrderDetail
(
order_id int CONSTRAINT PK_Order_Id  PRIMARY KEY  IDENTITY(1,1),
unit_price money NULL,
order_qty smallint NULL
      CONSTRAINT   CK_Order_Order_qty  CHECK(order_qty > 5),
total_qty AS (unit_price * order_qty),    /* пример вычисляемого столбца */
rowguid uniqueidentifier ROWGUIDCOL NOT NULL
      CONSTRAINT DF_Order_rowguid DEFAULT (NEWID()),
my_user varchar(20) DEFAULT USER, /* имя пользователя, вставившего строку */
)
```

Пример удаления таблицы после проверки ее существования (IF EXISTS):

```
DROP TABLE IF EXISTS Production.[OrderDetail]
```

Задание

Необходимо создать средствами T-SQL три обычных таблицы и три таблицы, обеспечивающие возможность сохранения копий строк из обычных таблиц при удалении данных. При создании обычных таблиц необходимо реализовать все изученные механизмы управления значениями столбцов.

Содержание отчета: тема и цель работы; прокомментированный SQL-код выполнения задания.

Контрольные вопросы

1 Перечислить этапы проектирования структуры таблиц.

2 Охарактеризовать основные символы нотации Бэкуса – Наура.

3 С помощью каких команд T-SQL можно создать или удалить таблицу?

4 Охарактеризовать механизмы управления значениями столбцов (PRIMARY KEY, FOREIGN KEY, UNIQUE, CHECK, DEFAULTS, NULL).

5 Хранится ли в таблице значение вычисляемого столбца?

6 Указать способы обеспечения уникальности значений в столбце таблицы.

7 Для чего нужна схема базы данных? Как создать новую схему базы данных? К какой схеме относятся по умолчанию создаваемые таблицы?

8 Из каких частей в SSMS состоит имя таблицы?

9 Когда применяются регулярный идентификатор имени объекта и идентификатор с разделителем (разделитель – это квадратные скобки ([ ]) или одинарные кавычки (' '))?

12 Изменение таблиц средствами SQL

Цель: изучить изменение структуры таблиц средствами T-SQL.

Теоретические положения

Для изменения структуры таблицы средствами T-SQL предусмотрен специальный оператор [4, 11]:

```
ALTER TABLE { database_name.schema_name.table_name | schema_name.table_name | table_name }
    {
    /* изменить определение уже существующего столбца*/
       ALTER COLUMN column_name
       {
           new_data_type  [ (precision [, scale]) ]
           [ NULL | NOT NULL ]
           | { ADD | DROP }
           { ROWGUIDCOL | PERSISTED | NOT FOR REPLICATION }
       }
    |  [ WITH CHECK | WITH NOCHECK ]
    /* добавить в таблицу новый столбец, вычисляемый столбец, ограничение на уровне таблицы */
         | ADD
         {
              <column_definition>
           | <computed_column_definition>
           | <table_constraint>
```

```
/* удалить из таблицы столбец, ограничение */
        | DROP
         [ {
            [ CONSTRAINT ][ IF EXISTS ]
            {   constraint_name  } [ ,...n ]
             | COLUMN [ IF EXISTS ]
             {  column_name   } [ ,...n ]
         } [ ,...n ]
        | [ WITH  CHECK | WITH NOCHECK  ] { CHECK | NOCHECK } CON-
STRAINT
            { ALL | constraint_name [ ,...n ] }
        | { ENABLE | DISABLE } TRIGGER
            { ALL | trigger_name [ ,...n ] }
```

С помощью оператора ALTER TABLE можно изменить определение уже имеющихся столбцов, удалить любой из них, добавить в таблицу новые столбцы.

Изменение определения столбца. Данная операция осуществляется с использованием предложения ALTER COLUMN, после которого помещается имя изменяемого столбца (column_name). Можно изменить тип данных столбца (new_data_type), размерность (precision) и точность (scale). Можно указать, разрешено ли столбцу содержать значения NULL. В этом случае обязательно нужно указать тип данных для столбца, даже если не планируется его изменять (просто указывается существующий тип данных). Если определяется для столбца свойство NOT NULL, необходимо, чтобы на момент изменения столбец не содержал ни одного значения NULL.

Нельзя изменить тип данных для столбца, который входит в состав первичного или внешнего ключей или является частью индекса. Не поддерживается изменение столбцов следующих типов: столбец типа данных timestamp, свойство ROWGUIDCOL для таблицы, а также вычисляемый столбец или используемый в вычисляемом столбце.

Добавление в таблицу нового столбца. Для определения нового столбца необходимо использовать ключевое слово ADD. За ним следует описание столбца, которое имеет такой же формат, как и при создании столбца с помощью оператора CREATE TABLE. Здесь же можно наложить на таблицу новые ограничения на значения столбцов.

Управление ограничениями на значения колонок. При указании ключевого слова WITH CHECK системе предписывается при добавлении новых ограничений на значения столбцов FOREIGN KEY или CHECK осуществлять проверку данных в таблице на соответствие этим ограничениям. По умолчанию данная проверка проводится для всех вновь создаваемых ограничений. Когда выполнение подобной проверки не требуется, используется WITH NOCHECK.

Отключение конкретного ограничения (NOCHECK CONSTRAINT) означает, что при вводе новых строк данные не будут проверяться на соответствие этому ограничению. Когда снова потребуется сделать ограничение активным, используется ключевое слово CHECK CONSTRAINT. При необходимости

можно управлять всеми ограничениями сразу с помощью ключевого слова ALL. Вариант WITH NOCHECK не рекомендуется использовать, за редкими исключениями. Любые нарушения ограничения, подавленные условием WITH NOCHECK во время добавления ограничения, могут привести к сбою будущих обновлений, если новые значения строк не соответствуют этому ограничению.

Удаление столбцов из таблицы. В случае необходимости можно удалить из таблицы некоторые столбцы. Для этого используется ключевое слово DROP. Можно удалить как конкретный столбец (ключевое слово COLUMN), так и определенное ограничение на значение столбца. Однако нельзя удалять следующие столбцы: столбцы, задействованные в индексе; столбцы, полученные в результате репликации; столбцы, для которых определены любые ограничения на значения (например, первичного или внешнего ключа, CHECK); столбцы, для которых определены значения по умолчанию.

Управление триггерами. Ключевое слово DISABLE TRIGGER отключает триггер. При этом в процессе изменения данных в таблице те действия, которые определены в триггере как реакция системы на эти изменения, не производятся, хотя триггер продолжает существовать. Чтобы активизировать триггер, необходимо использовать команду с ключевым словом ENABLE TRIGGER. Если требуется управлять сразу всеми триггерами, используется ключевое слово ALL.

В следующем примере к столбцу в таблице добавляется ограничение. В столбце имеется значение, нарушающее это ограничение. Поэтому во избежание проверки ограничения относительно существующих строк, а также для того, чтобы разрешить добавление ограничения, применяется WITH NOCHECK.

```
CREATE TABLE dbo.example (column_a  int);
GO
INSERT INTO dbo.example VALUES (-1) ;
GO
ALTER TABLE dbo.example WITH NOCHECK
ADD CONSTRAINT ex_check CHECK (column_a > 1) ;
GO
```

Задание

Необходимо изменить структуру таблиц, созданных в лабораторной работе № 10, так, чтобы в них можно было хранить информацию о времени удаления данных из таблицы.

Содержание отчета: тема и цель работы; прокомментированный SQL-код выполнения задания.

Контрольные вопросы

1 Синтаксис команды изменения структуры таблицы средствами T-SQL.
2 Как изменить определение столбца таблицы?
3 Как добавить новый столбец или удалить столбец из таблицы?

4 С помощью каких предложений производится управление ограничениями на значения столбцов и триггерами?

13 Создание связей между таблицами средствами SQL

Цель: научиться создавать связи между таблицами средствами T-SQL.

Теоретические положения

Для обеспечения связей 1:1 и 1:M необходимо, чтобы одна из таблиц содержала ссылку (внешний ключ) на вторую. Внешний ключ – это столбец (или группа столбцов таблицы), содержащий значения, совпадающие со значениями первичного ключа в этой же или другой таблице.

Синтаксис предложения FOREIGN KEY следующий:

```
[CONSTRAINT c_name]
[[FOREIGN KEY] ({col_name1} ,...)]
REFERENCES table_name ({col_name2},...)
[ON DELETE {NO ACTION | CASCADE | SET NULL | SET DEFAULT}]
[ON UPDATE {NO ACTION | CASCADE | SET NULL | SET DEFAULT}]
```

Предложение FOREIGN KEY явно определяет все столбцы, входящие во внешний ключ. В предложении REFERENCES указывается имя таблицы, содержащей столбцы, создающие соответствующий первичный ключ. Количество столбцов и их тип данных в предложении FOREIGN KEY должны совпадать с количеством соответствующих столбцов и их типом данных в предложении REFERENCES (и, конечно же, они должны совпадать с количеством столбцов и типами данных в первичном ключе таблицы, на которую они ссылаются).

Таблица, содержащая внешний ключ, называется ссылающейся (или дочерней), а таблица, содержащая соответствующий первичный ключ, – ссылочной или родительской.

Например, создадим две таблицы, связанные отношением 1:1. Столбец внешнего ключа id_a_link таблицы TableB нужно сделать уникальным. Это гарантирует, что в ячейке столбца FOREIGN KEY в таблице TableB может быть только одно значение, которое соответствует такому же значению в столбце PRIMARY KEY в таблице TableA (т. е. одна строка в таблице TableB связана с одной строкой в таблице TableA).

```
CREATE TABLE TableB
( Id INT PRIMARY KEY IDENTITY(1,1),
  Name VARCHAR(255))

CREATE TABLE TableA
( Id INT PRIMARY KEY IDENTITY(1,1),
```

```
Name VARCHAR(255),
TableBRelation  INT UNIQUE,
FOREIGN KEY (TableBRelation) REFERENCES TableB (Id))
```

Задание

Необходимо средствами T-SQL установить отношения между резервными таблицами из лабораторной работы № 12, соответствующие отношениям между основными таблицами. Обоснование реализованных типов связей необходимо представить в виде таблицы 13.1.

Таблица 13.1 – Обоснование реализованных типов связей

Наименование типа созданной связи	SQL-определение столбцов, участвующих в связи со стороны родительской таблицы	SQL-определение столбцов, участвующих в связи со стороны дочерней таблицы	SQL-определение созданной связи

Содержание отчета: тема и цель работы; подробно прокомментированный SQL-код выполнения задания.

Контрольные вопросы

1 Какие типы связей можно реализовать в MS SQL Server?
2 Для чего предназначены предложения FOREIGN KEY и REFERENCES?
3 Как создаются связи 1:1, 1:M, M:M?
4 Как создаются идентифицирующая, неидентифицирующая обязательная и необязательная связи?

14 Создание sql-скрипта заполнения базы данных

Цель: получить навыки создания sql-скриптов заполнения БД.

Теоретические положения

Сценарий (скрипт) – последовательность операторов T-SQL. Для подготовки сценария, отладки и выполнения используется SSMS.

Автоматическая генерация скриптов из обозревателя объектов.

Для таблицы можно сгенерировать скрипты для создания таблицы, удаления таблицы, выборки данных из таблицы, скрипты для добавления новых данных в таблицу, а также для изменения и удаления существующих записей.

Для генерации скрипта таблицы текущей БД из контекстного меню выбирается команда «Создать сценарий для таблицы» → «Используя CREATE».

Для генерации скрипта БД после выделения имени БД из контекстного меню выбирается команда «Задачи» → «Сформировать скрипты» → «Создать скрипт для всей базы данных и всех ее объектов» → «Указание порядка сохранения скриптов: Открыть в новом окне запроса» → «Дополнительные параметры создания скрипта: Типы данных для внесения в скрипт – Схема и данные».

При запуске SSMS и подключении к SQL-серверу по умолчанию выполняется команда USE master, т. е. работа идет с системной БД master. Для выбора иной БД следует выполнить команду USE Имя_Базы_Данных. Перед запуском на исполнение сгенерированного скрипта БД аналогичным образом в первой строке скрипта нужно указывать USE Имя_Базы_Данных.

Для создания скрипта для административных операций (например, резервное копирование базы данных, создание учетной записи и т. д.) можно воспользоваться контекстным меню «Задачи» → «Создать резервную копию…» → «Скрипт», что позволит автоматически создать скрипт, в который будут подставлены введенные в полях формы на экране значения.

Задание

Необходимо, используя команды INSERT, внести в базу данных не менее 200 записей. Пример команды INSERT:

```
INSERT INTO Table1(t_id, product, date) VALUES (3, 'Коробка', 2020-12-26)
```

Для заполненной БД сгенерировать скрипт, содержащий схему базы данных и данные в таблицах.

Содержание отчета: тема и цель работы; скрипт, содержащий схему БД и данные.

Контрольные вопросы

1 Что такое скрипт (сценарий) и для решения каких задач он используется?
2 Какие виды скриптов существуют?
3 Для чего предназначен оператор GO [4, с. 108–109]?
4 Перечислить основные характеристики базы данных [4, с. 153–162].

15 Язык SQL. Добавление, изменение и удаление данных в таблицах средствами SQL

Цель: научиться добавлять, изменять и удалять данные в таблицах с помощью операторов T-SQL.

Теоретические положения

Для вставки данных в таблицу средствами T-SQL используется оператор INSERT, имеющий следующий синтаксис [3, 4, 11]:

```
INSERT [INTO] { Table_name | View_name }
{ [ (column_1, …, column_n) ]
{  VALUES
( {   DEFAULT  |  NULL  |  expression } [,... n ] )
      | derived_table  |  execute_statement  }
|  DEFAULT VALUES
}
```

Ключевое слово INSERT и необязательное ключевое слово INTO вводят инструкцию. Аргумент Table_name задает целевую таблицу, в которую необходимо вставить данные. Можно в качестве цели вставки задать имя представления. Далее может указываться список столбцов, разделенных запятыми, который будет получать вставляемые данные. Вставляемые значения можно задать ключевыми словами DEFAULT, NULL или выражениями. В качестве альтернативы можно использовать инструкцию SELECT для создания временной таблицы, которая станет поставщиком данных для вставки.

derived_table – это любая допустимая инструкция SELECT, возвращающая строки данных, которые загружаются в таблицу.

execute_statement – любая допустимая инструкция EXECUTE, возвращающая данные с помощью инструкций SELECT.

Для создания набора вставляемых данных можно использовать инструкцию выполнения вместе с хранимой процедурой или пакетом SQL. Предложение DEFAULT VALUES использует значения таблицы по умолчанию для каждого столбца в новой строке. Инструкция INSERT может вставлять множество строк данных, если в качестве поставщика данных используется временная таблица или результаты инструкции выполнения.

При ссылке на типы данных символов Unicode nchar, nvarchar и ntext выражение 'expression' должно начинаться с заглавной буквы 'N'. Если префикс «N» не указан, SQL Server выполнит преобразование строки в кодовую страницу, соответствующую параметрам сортировки базы данных или столбца, действующим по умолчанию.

Если в явном виде не указан список столбцов таблицы, то инструкция INSERT будет пытаться вставить значения в каждый столбец таблицы в порядке предоставления значений.

Если выполняется вставка данных в представление, а не таблицу, то представление должно быть обновляемым. Кроме того, с помощью одной инструкции INSERT можно вставить данные только в одну из базовых таблиц, на которые ссылается представление.

Если таблица имеет триггер INSTEAD OF INSERT, то вместо любой инструкции INSERT, пытающейся поместить строки в эту таблицу, будет выполняться код в триггере [4].

Если столбец имеет свойство IDENTITY (столбец идентификаторов, счетчик), при вставке строки имя этого столбца и значение поля для этого столбца в команде INSERT не указывают. Для такого столбца сервер автоматически вычисляет новое значение. Оператор SET IDENTITY_INSERT < имя таблицы > { ON | OFF } отключает (значение ON) или включает (OFF) использование автоинкремента. И, используя оператор SET IDENTITY_INSERT, можно явным образом задать требуемое значение для ячейки в столбце идентификаторов:

```
SET  IDENTITY_INSERT  Product ON;
INSERT  INTO  Product(product_id, name, value)
VALUES  (17285, 'Сахар', 10')
```

Еще один способ вставки данных во вновь создаваемую таблицу – с помощью **инструкции SELECT INTO**, которая является вариацией простой инструкции SELECT. Схематически ее синтаксис выглядит так [4]:

```
SELECT Select_List
INTO имя_новой_таблицы
FROM исходная_таблица
[WHERE условие]
[GROUP BY выражение]
[HAVING условие]
[ORDER BY выражение]
```

Инструкция SELECT INTO в целом идентична инструкции SELECT. Новым элементом является предложение INTO. В нем можно задать имя таблицы (с помощью любого корректного идентификатора SQL Server), и инструкция SELECT INTO создаст эту таблицу. Таблица будет иметь по одному столбцу для каждого столбца результатов выполнения инструкции SELECT. Имена и типы данных этих столбцов будут такими же, как и у соответствующих столбцов в списке SELECT. Другими словами, инструкция SELECT INTO использует результаты выполнения инструкции SELECT и преобразует их в постоянную таблицу [4].

В таблицах, создаваемых с помощью инструкции SELECT INTO, не содержатся индексы, первичные ключи, внешние ключи, значения по умолчания и триггеры. Если для таблицы необходимо создать какой-либо из вышеперечисленных объектов, следует создать таблицу с помощью инструкции CREATE TABLE, а затем использовать инструкцию INSERT для заполнения таблицы данными. Обычно это сделать проще, чем создавать таблицу с помощью инструкции

SELECT INTO, а затем фиксировать другие свойства с помощью инструкции ALTER TABLE [4].

Инструкцию SELECT INTO можно также использовать для создания временной таблицы. При этом первым символом в имени таблицы следует поставить знак решетки (#). Временные таблицы полезны при работе с SQL в длинном триггере или хранимой процедуре, когда во время выполнения процедуры нужно отслеживать информацию. SQL Server автоматически удаляет временные таблицы после окончания работы с ними.

Инструкция UPDATE реализует один из способов изменения любых данных, содержащихся в таблице. Можно написать инструкцию UPDATE таким образом, чтобы она влияла только на отдельное поле в отдельной строке либо вычисляла изменения в столбце во всех строках таблицы. Можно также написать инструкцию, которая будет изменять все строки из множества столбцов. Синтаксис инструкции UPDATE:

```
UPDATE { Table_name | View_name }
SET { имя_столбца = {выражение | DEFAULT | NULL}
| @ переменная = выражение | @переменная-столбец = выражение
{ [ FROM {исходная_таблица} [,... n] ]
[ WHERE условие_поиска ] }
```

Для обновляемых строк нужно задать имя таблицы или представления. Ключевое слово SET представляет вносимые изменения. Можно задать значение столбца равным выражению, значению по умолчанию или пустому значению NULL. Можно присвоить выражение локальной переменной. Можно объединить присвоение значения локальной переменной и столбцу в одном и том же выражении. В одной директиве SET можно задать множество столбцов.

Самая простая инструкция UPDATE выполняет одно изменение, которое влияет на все строки таблицы. Например, можно изменить стоимость всех продуктов, перечисленных в таблице «Продукция», на 20 денежных единиц с помощью следующей инструкции:

```
UPDATE Продукция SET Стоимость = 20.00
```

Инструкцию UPDATE можно также использовать для присвоения значений локальным переменным. Например, следующий пакет создает локальную переменную, присваивает ей значение и выводит результат на печать:

```
DECLARE @Name nvarchar(50) UPDATE Продукция
SET @Наименование = Крем
PRINT @Наименование
```

В данном случае SQL Server обработал все строки в таблице, хотя UPDATE не изменяет в ней никаких данных. Чтобы обновление было более эффективным, можно добавить директиву WHERE, отбирающую отдельную запись:

```
DECLARE @Name nvarchar(50) UPDATE Продукция
SET @Наименование= Крем
WHERE ПродукцияID = 418
PRINT @Наименование
```

Если инструкция UPDATE не отберет никаких строк, она не присвоит значение локальной переменной. Чтобы присвоить значение локальной переменной, не ссылаясь на таблицу, используется инструкция SET.

Операция обновления столбца со свойством IDENTITY представляет собой комбинацию инструкции DELETE с удалением ненужного значения из ячейки столбца идентификаторов и инструкции INSERT со вставкой требуемого значения в ячейку столбца идентификаторов.

Для удаления записей при помощи запросов из существующей таблицы можно использовать **инструкцию DELETE**. Ее базовый синтаксис следующий:

```
DELETE [FROM]
{ Table_name | View_name }
[ FROM исходная_таблица ]  [ WHERE условия_поиска ]
```

Необязательное ключевое слово FROM можно использовать для удобочитаемости инструкции SQL. Если не включить предложение WHERE в инструкцию DELETE, то инструкция удалит все строки из целевой таблицы [4].

Простейшая инструкция DELETE удаляет все строки из целевой таблицы. Во избежание нежелательных последствий с помощью инструкции SELECT INTO создается копия таблицы «Клиенты» и выполняется работа уже с копией:

```
SELECT * INTO КлиентыCopy FROM Клиенты
DELETE FROM КлиентыCopy
DROP TABLE КлиентыCopy
```

Инструкция DROP TABLE удаляет временную копию таблицы после выполнения первых двух инструкций, так что для следующего примера необходимо создать новую копию.

С помощью ограничивающего предложения WHERE можно удалить множество строк, но не целую таблицу:

```
SELECT * INTO КлиентыCopy FROM Клиенты
DELETE КлиентыCopy WHERE Фамилия LIKE 'A%'
DROP TABLE КлиентыCopy
```

Инструкция DELETE не может удалять из таблицы строки со значениями NULL на пассивной стороне внешнего объединения. Рассмотрим, к примеру, инструкцию DELETE со следующей директивой FROM [4].

```
FROM  Клиенты
LEFT JOIN Заказ ON Клиенты.КонтактID = Заказ.КонтактID
```

В данном случае таблица «Заказ» содержит значения NULL. Это означает, что столбцы из этой таблицы будут содержать значения NULL для строк, соответствующих контактам, для которых не размещены заказы. В таком случае можно использовать инструкцию DELETE только для удаления строк из таблицы «Клиенты», но не для удаления строк из таблицы «Заказы».

Если инструкция DELETE пытается нарушить работу триггера или ограничения, поддерживающего целостность ссылок, она не будет выполнена. Даже если только одна удаляемая строка из набора нарушает ограничивающие условия, то выполнение инструкции будет отменено, а SQL Server вернет сообщение об ошибке и не станет удалять строки [4].

При выполнении инструкции DELETE для таблицы, в которой определен триггер INSTEAD OF DELETE, сама инструкция DELETE не будет выполнена. Вместо этого будут выполняться действия триггера для каждой удаляемой строки в таблице [4].

Для удаления первых трех записей в таблице, отсортированной в алфавитном порядке, можно использовать следующую инструкцию:

```
SELECT * INTO КлиентыCopy FROM Клиенты DELETE КлиентыCopy
FROM (SELECT TOP 3 * FROM КлиентыCopy ORDER BY Фамилия)
AS t
WHERE КлиентыCopy.КонтактID = t.КлиентID
DROP TABLE КлиентыCopy
```

Здесь инструкция SELECT в круглых скобках является подчиненным запросом, возвращающим базовый набор строк для инструкции DELETE. Результату этого подчиненного запроса присваивается псевдоним t, а директива WHERE задает параметры сравнения строк из t с перманентной таблицей. Затем директива DELETE автоматически удаляет все совпавшие строки.

Инструкция TRUNCATE TABLE ИмяЦелевойТаблицы удаляет все строки в целевой таблице, не записывая в журнал транзакций удаление отдельных строк, за счет чего выполняется быстрее и требует меньших ресурсов системы.

Далее будут рассмотрены некоторые способы отбора записей в запросах.

Оператор IN позволяет определить набор значений, которые должны иметь столбцы: WHERE выражение [NOT] IN (выражение). Выражение в скобках после IN определяет набор значений. Этот набор может вычисляться динамически на основании, например, еще одного запроса либо это могут быть константы.

Например, выберем книги издательств Amedia, либо Aquarius, либо BHV:

```
SELECT * FROM Book
WHERE publisher IN ('Amedia', 'Aquarius', 'BHV')
```

Оператор BETWEEN определяет диапазон значений с помощью начального и конечного значения, которому должно соответствовать выражение: WHERE выражение [NOT] BETWEEN начальное_значение AND конечное_значение.

В таблице 15.1 представлены допустимые подстановочные символы, их значения и примеры использования.

Таблица 15.1 – Подстановочные символы, используемые в шаблонах LIKE

Подстановочный знак	Значение	Пример	Результат
%	Символ-шаблон, заменяющий любую последовательность символов	WHERE [title] LIKE 's%'	Строка, начинающаяся с s: Samantha или sven
_ (подчеркивание)	Символ-шаблон, заменяющий любой одиночный символ	WHERE [titleofcourtesy] LIKE 'm_.'	Ms. Mr.
[<список символов>]	Один символ из списка	WHERE [firstname] LIKE '[sp]%'	Строка, в которой первый символ s или p: Sara или Paul
[<диапазон символов>]	Один символ из диапазона. При этом можно перечислить сразу несколько диапазонов (например, [0-9a-z])	WHERE [freight] LIKE '6[5-7]%'	Строка, в которой второй символ – цифра 5, 6 или 7: 65,83 или 677,54
[^<список или диапазон символов>]	В сочетании с квадратными скобками исключает из поискового образца символы из списка или диапазона	WHERE [shipaddress] LIKE '[^0-9]%'	Строка, в которой первый символ не цифра
ESCAPE ' '	Для поиска символа, который является подстановочным знаком, его указывают после Escape-символа с помощью ключевого слова ESCAPE	WHERE col1 LIKE '!_%' ESCAPE '!'	Поиск строки, которая начинается со знака подчеркивания (_), используя в качестве Escape-символа (!)
'ymd'	Для поиска даты используется форма без разделителей, которая не зависит от языка входа в систему для всех типов данных даты и времени	WHERE [birthdate] = '19581208'	1958-12-08 00:00:00.000

Агрегатные функции выполняют вычисления над значениями в наборе строк. Все агрегатные функции, за исключением COUNT(*), игнорируют значения NULL (таблица 15.2).

Выражение в функциях AVG и SUM должно представлять числовое значение. Выражение в функциях MIN, MAX и COUNT может представлять числовое или строковое значение или дату.

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

Таблица 15.2 – Агрегатные функции

Синтаксис	Назначение
AVG()	Вычисляет среднее значение для указанного столбца
SUM()	Суммирует все значения в указанном столбце
MIN()	Находит минимальное значение в указанном столбце
MAX()	Находит максимальное значение в указанном столбце
COUNT()	Подсчитывает количество строк с непустым (не NULL) значением указанного столбца
COUNT(*)	Подсчитывает общее количество строк, удовлетворяющих условию, включая пустые (NULL)

Задание

Необходимо для разрабатываемой БД написать для каждой таблицы по три различных команды INSERT, UPDATE, DELETE с использованием различных функций (не менее 15 различных функций) и предикатов в условии отбора. Перечень функций T-SQL содержится в [3, 4, 11].

Содержание отчета: тема и цель работы; прокомментированный SQL-код выполнения задания.

Контрольные вопросы

1 Объяснить синтаксис операторов INSERT, SELECT INTO, UPDATE, DELETE и указать ограничения для данных операторов.

2 Привести примеры использования INSERT, UPDATE, DELETE.

3 Охарактеризовать особенности использования операторов INSERT, UPDATE, DELETE по отношению к столбцу идентификаторов IDENTITY.

4 Указать назначение и ограничения оператора TRUNCATE TABLE.

5 Перечислить подстановочные символы, используемые в шаблонах LIKE.

6 Перечислить основные группы функций T-SQL и охарактеризовать наиболее часто используемые функции.

16 Язык SQL. Работа с представлениями

Цель: научиться создавать представления в MS SQL Server средствами Transact-SQL.

Теоретические положения

Представление – это именованный запрос на выборку, сохраненный в базе данных, который выглядит и работает как таблица, при обращении по имени создает виртуальную таблицу, наполняя ее актуальными данными из БД, с которой

можно работать так же, как с реально существующей на диске таблицей. Физически представление реализовано в виде SQL-запроса, на основе которого производится выборка данных из одной или нескольких таблиц или представлений. Представление часто применяется для ограничения доступа пользователей к конфиденциальным данным в таблице [3, 4, 11].

Для создания представлений средствами Transact-SQL используется следующая конструкция:

```
CREATE VIEW [ schema_name . ] view_name [ (column [ ,...n ] )
[ WITH { ENCRYPTION | SCHEMABINDING | VIEW_METADATA } ]
AS
select_statement
[WITH CHECK OPTION]
```

Рассмотрим составляющие данной конструкции.

view_name – имя представления. При указании имени необходимо придерживаться тех же правил и ограничений, что и при создании таблицы.

column – имя столбца, которое будет использоваться в представлении (длина имени до 128 символов). Имена столбцов перечисляются через запятую в соответствии с их порядком в представлении. Имена столбцов можно указывать в команде SELECT, определяющей представление.

WITH ENCRYPTION – данный параметр предписывает серверу шифровать код SQL-запроса. Это гарантирует, что пользователи не смогут просмотреть код запроса и использовать его. Если при определении представления необходимо скрыть имена исходных таблиц и колонок, а также алгоритм объединения данных, то следует использовать эту опцию.

WITH SCHEMABINDING – привязывает представление к схеме базовой таблицы. Нельзя будет изменить описание базовых таблиц, если это повлияет на представление. Сначала нужно будет изменить или удалить само представление для сброса зависимостей от таблицы, которую требуется изменить. При указании этого атрибута базовые таблицы в представлении должны быть записаны в виде <схема>.<таблица>.

WITH VIEW_METADATA – указывает, что в API-интерфейсы DB-Library, ODBC и OLE DB экземпляр MS SQL Server возвратит сведения о метаданных представления, а не о базовых таблицах этого представления.

select_statement – код запроса SELECT, выполняющий выборку, объединение и фильтрацию строк из исходных таблиц и представлений. Можно использовать команду SELECT любой сложности со следующими ограничениями:

1) нельзя создавать новую таблицу на основе результатов, полученных в ходе выполнения запроса, т. е. запрещается использование параметра INTO;

2) нельзя проводить выборку данных из временных таблиц, т. е. нельзя использовать имена таблиц, начинающихся на # или ##;

3) в представление нельзя включать предложение ORDER BY, если только в списке выбора инструкции SELECT нет также предложения TOP.

Предложение WITH CHECK OPTION применяется только к обновлениям, выполненным через представление. Оно неприменимо к обновлениям, выполненным непосредственно в базовых таблицах представления. Если имеется представление с ограничением фильтра в предложении WHERE инструкции SELECT, а затем с помощью представления были изменены строки таблицы, то можно изменить некоторое значение так, что задействованная строка уже не будет удовлетворять фильтру предложения WHERE. Возможно даже обновление строк, которые выходят за пределы области фильтра. Предложение WITH CHECK OPTION препятствует подобному исчезновению строк при обновлении через представление, а также ограничивает модификации только строками, которые удовлетворяют критериям фильтра.

Чтобы выполнить представление, т. е. получить данные в виде виртуальной таблицы, необходимо выполнить запрос SELECT к представлению так же, как и к обычной таблице: SELECT * FROM view_name.

Для удаления представления используется команда T-SQL DROP VIEW{view […n]}. За один раз можно удалить несколько представлений.

Обновляемые представления не содержат функции агрегирования или вычисляемые столбцы. Кроме того, представление, указанное в предложении FROM инструкции DELETE, должно содержать ровно одну таблицу (имеется в виду предложение FROM, используемое для создания представления, а не директива FROM в инструкции DELETE) [4].

В качестве примера приведено представление, предоставляющее информацию об экземплярах книг, которые были изданы за последние пять лет.

```
CREATE VIEW Get_full_info_ copy_of_book
AS
SELECT        /*Указываем, какие поля будут выбраны*/
Copy_of_book.cb_code_of_copy,
Book.b_author_last_name,        Book.b_title,        Book.b_year_of_publication,
Book.b_publisher,
Copy_of_book.cb_d_number,
Copy_of_book.cb_mark_of_write_off, Copy_of_book.cb_mark_of_replacement
FROM          /*Указываем таблицу и связанные с ней таблицы, из которых выбираются связанные данные*/
Book
    INNER JOIN Copy_of_book
    ON Book.b_id_book = Copy_of_book.cb_b_id_book
WHERE  YEAR(Book.b_year_of_publication)  BETWEEN  YEAR(GETDATE()-5)  AND  YEAR(GETDATE())
/* GETDATE()  возвращает текущую дату, YEAR(<дата>) – год <даты> */
```

Задание

В разрабатываемой базе данных необходимо реализовать 15 представлений с использованием стандартных функций T-SQL. При этом должно быть использовано не менее 10 различных функций.

Содержание отчета: тема и цель работы; SQL-код 15 представлений.

Контрольные вопросы

1 Что такое представление и в каких случаях целесообразно его использовать? Перечислить способы создания представлений.

2 Какие виды представлений различают? Что такое обновляемые представления? Что такое кэширующие (материализованные, индексированные) представления?

3 Перечислить операторы SQL, с помощью которых представления создаются, удаляются и изменяются.

4 Перечислить ограничения при создании представлений.

17 Создание индексов средствами языка SQL

Цель: получить навыки определения индексируемых столбцов таблиц.

Теоретические положения

Индекс представляет собой дополнение к таблице, помогающее ускорить поиск необходимых данных за счет физического или логического их упорядочивания. Он является набором ссылок, упорядоченным по определенному (индексируемому) столбцу таблицы. Физически индекс представляет собой упорядоченный набор значений из индексированного столбца с указателями на места физического размещения исходных строк в структуре базы данных. В индексе хранится не информация обо всей строке данных, а лишь ссылка на нее. Использование индексов позволяет избежать полного сканирования таблицы. На сегодняшний день существует большое количество разновидностей индексов, основными из которых являются кластерный, некластерный, уникальный, покрывающий, полнотекстовый. Более подробно они описаны в [4, 11].

При выборе столбца для индекса следует проанализировать, какие типы запросов чаще всего выполняются и какие столбцы являются ключевыми.

Ключевые столбцы – это такие колонки, которые задают критерии выборки данных, например порядок сортировки. Не стоит индексировать столбцы, которые только считываются и не играют никакой роли в определении порядка выполнения запроса. Не следует индексировать слишком длинные колонки, например колонки с адресами или названиями компаний, достигающие длины несколько десятков символов. В крайнем случае, можно создать укороченный вариант такого столбца, выбрав из него до десяти первых символов, и индексировать его. Индексирование длинных столбцов может существенно снизить производительность работы сервера. Индекс является самостоятельным объектом БД, но он связан с определенным столбцом таблицы.

Индексы создаются в следующих случаях:

– если операции чтения из таблицы выполняются гораздо чаще, чем операции модификации;

– если поле (или совокупность полей) часто используется в запросах в разделе WHERE;

– если нужно обеспечить уникальность значения поля (или совокупности полей), не являющегося первичным ключом (в этом случае создается уникальный индекс);

– если поле (или совокупность полей) является внешним ключом (т. к. в таком случае индексы могут существенно ускорить выполнение JOIN-запросов).

Формат команды CREATE INDEX на T-SQL имеет вид:

```
CREATE [UNIQUE] [CLUSTERED | NONCLUSTERED]
INDEX index_name
ON table_or_view_name ( column [...n] )
[ INCLUDE ( column_name [ ,...n ] ) ]
[ WITH { PAD_INDEX = { ON | OFF } | FILLFACTOR = f
    | IGNORE_DUP_KEY } [ ,...n ] ) ]
```

При указании ключевого слова UNIQUE будет создан уникальный индекс, гарантирующий уникальность значений в индексируемом столбце. При создании такого индекса сервер выполняет предварительную проверку столбца на уникальность значений. Если в столбце есть хотя бы два одинаковых значения, индекс не создается. В индексируемом столбце также желательно запретить хранение значений NULL. После того как для столбца создан уникальный индекс, сервер не разрешает выполнение команд INSERT и UPDATE, которые приведут к появлению дублирующихся значений. Уникальный индекс является своеобразной надстройкой и может быть реализован как для кластерного, так и для некластерного индексов. В одной таблице могут существовать один уникальный кластерный индекс и множество уникальных некластерных индексов.

CLUSTERED – создаваемый индекс будет кластерным, т. е. при его определении в таблице физическое расположение данных перестраивается в соответствии со структурой индекса. Информация об индексе и сами данные физически располагаются вместе. Кластерным может быть только один индекс в таблице. Кластерный индекс создается до создания любых некластерных, т. к. при создании кластерного индекса все существующие некластерные индексы таблицы перестраиваются. Если аргумент CLUSTERED не указан, создается некластерный индекс. В качестве кластерного индекса следует выбирать наиболее часто используемые столбцы. Следует избегать создания кластерного индекса для часто изменяемых столбцов, т. к. сервер должен будет выполнять физическое перемещение всех данных в таблице, чтобы они находились в упорядоченном состоянии, как того требует кластерный индекс.

NONCLUSTERED – создаваемый индекс будет некластерным, и, в отличие от кластерных, он не перестраивает физическую структуру таблицы, а лишь организует ссылки на соответствующие строки. Каждая таблица может содержать

до 999 некластерных индексов независимо от способа их создания: неявно с помощью ограничений PRIMARY KEY и UNIQUE или явно с помощью инструкции CREATE INDEX. Однако лучше ограничиться 4–5 индексами.

index_name – имя индекса, по которому он будет распознаваться командами Transact-SQL. Имя индекса должно быть уникальным в пределах таблицы.

table_or_view_name(column [...n]) – имя таблицы или представления, в которой содержатся один или несколько индексируемых столбцов. В скобках указываются имена столбцов, на основе которых будет построен индекс. Не допускается построение индекса на основе столбцов с типом данных ntext, text, varchar(max), nvarchar(max), varbinary(max), xml или image. Если указывается несколько столбцов, то создаваемый индекс будет составным. В один составной индекс можно включить до 16 столбцов.

Параметр INCLUDE позволяет создать покрывающий индекс, т. е. некластерный индекс, содержащий в своих листовых узлах информацию из дополнительного неключевого поля, не использующегося при создании самого индекса.

В предложении WITH перечисляются параметры создаваемого индекса.

Коэффициент заполнения FILLFACTOR = f задает заполнение в процентах каждой страницы индекса во время его первоначального создания или пересоздания. Значение f можно установить в диапазоне от 1 до 100 (по умолчанию f = 0). Значения коэффициентов заполнения 0 и 100 идентичны. Если f равен 100, компонент Database Engine создает индексы с полностью заполненными страницами конечного уровня. Чем больше значение f, тем меньше свободного места в страницах листьев индекса. Например, при значении f = 60 каждая страница листьев индекса будет иметь 40 % свободного места для вставки строк индекса в дальнейшем. Производительность операций считывания снижается обратно пропорционально значению коэффициента заполнения. Для редко изменяемых таблиц рекомендуется принимать f = 100; для часто изменяемых таблиц f = 50–70; в случае промежуточной ситуации f = 80–90 [4].

Параметр PAD_INDEX = {ON | OFF} задает возможность использования разреженного индекса. ON – допустим разреженный индекс, OFF (значение по умолчанию) – недопустим. В случае разреженного индекса должен присутствовать и коэффициент заполнения FILLFACTOR, задающий процент разрежения.

Параметр IGNORE_DUP_KEY = { ON | OFF } определяет ответ на ошибку, случающуюся, когда операция вставки пытается вставить в уникальный индекс повторяющиеся значения ключа. Применяется только к операциям вставки, производимым после создания или перестроения индекса. Параметр не работает во время выполнения инструкции CREATE INDEX, ALTER INDEX или UPDATE. Значение по умолчанию – OFF, которое означает, что если в уникальный индекс вставляются повторяющиеся значения ключа, выводится сообщение об ошибке и будет выполнен откат всей операции INSERT. Значение ON означает, что если в уникальный индекс вставляются повторяющиеся значения ключа, выводится предупреждающее сообщение и с ошибкой завершаются только строки, нарушающие ограничение уникальности.

Для удаления индекса используется команда DROP INDEX, имеющая следующий синтаксис: DROP INDEX 'table.index' [...n]. Аргумент 'table.index' определяет удаляемый индекс в таблице.

Задание

Необходимо обосновать выбор колонок таблиц для создания индексов в разрабатываемой БД и создать 5 индексов для таблиц БД.

Содержание отчета: тема и цель работы; прокомментированный SQL-код выполнения задания.

Контрольные вопросы

1 Что такое кластерный, некластерный и уникальный индексы?

2 Какие критерии учитываются при выборе столбцов для индексирования?

3 С помощью каких команд T-SQL задаются различные виды индексов?

4 Перечислить общие рекомендации при планировании стратегии индексирования (назвать не менее 8 рекомендаций).

5 Когда использование индексов нецелесообразно?

18 Язык SQL. Создание хранимых процедур

Цель: научиться создавать хранимые процедуры в СУБД MS SQL Server с использованием команд T-SQL; реализовать хранимые процедуры для вставки, удаления, изменения данных.

Теоретические положения

Хранимая процедура – это скомпилированный набор SQL-предложений, сохраненный на сервере баз данных как именованный объект и выполняющийся как единый фрагмент кода. Хранимые процедуры могут принимать и возвращать параметры. При этом клиент осуществляет только вызов хранимой процедуры по ее имени, затем сервер базы данных выполняет блок команд, составляющих тело вызванной процедуры, и возвращает клиенту результат [3, 4, 11].

Хранимые процедуры обычно используются для поддержки ссылочной целостности данных и реализации бизнес-правил. В последнем случае повышается скорость разработки приложений, поскольку если бизнес-правила изменяются, можно изменить только текст хранимой процедуры, не изменяя клиентские приложения. По сравнению с обычными SQL-запросами, посылаемыми из клиентского приложения, они требуют меньше времени для подготовки к выполнению, поскольку скомпилированы и сохранены.

Хранимые процедуры имеют следующее определение:

```
CREATE { PROC | PROCEDURE } [schema_name.] procedure_name [;number]
[ {@parameter data_type} [= default] [ OUT | OUTPUT | [READONLY]] [,...n]
AS sql_statement [...n]
```

Рассмотрим составляющие данной конструкции.

procedure_name – имя создаваемой процедуры. Используя префиксы sp_, # и ##, можно определить создаваемую процедуру как системную или временную (локальную или глобальную). Имя процедуры должно характеризовать действие, выполняемое ею, и записываться в формате <глагол_объект>.

number – параметр определяет идентификационный номер хранимой процедуры, однозначно определяющий ее в группе процедур.

@parameter – определяет имя параметра, который будет использоваться создаваемой хранимой процедурой для передачи входных или выходных данных. Параметры, определяемые при создании хранимой процедуры, являются локальными переменными, поэтому несколько хранимых процедур могут иметь абсолютно идентичные параметры. Можно объявить один или несколько параметров, максимум – 2100.

data_type – определяет, к какому типу данных должны относиться значения параметра описываемой процедуры.

default – позволяет определить для параметра значение по умолчанию, которое хранимая процедура будет использовать в случае, если при ее вызове указанный параметр был опущен.

OUT | OUTPUT – определяет указанный параметр как выходной. Его значение может быть использовано вызвавшей программой.

READONLY – означает, что значение параметра не может быть изменено внутри хранимой процедуры.

AS – ключевое слово, определяющее начало кода хранимой процедуры. После этого ключевого слова следуют команды T-SQL, которые и составляют непосредственно тело процедуры (sql_statement). Здесь можно использовать любые команды, включая вызов других хранимых процедур, за исключением команд, начинающихся с ключевого слова CREATE.

Более подробно вопросы создания хранимых процедур различных видов рассмотрены в [4, 11].

Далее приведен пример хранимой процедуры, возвращающей количество экземпляров какой-либо книги.

```
/* проверяется, существует ли хранимая процедура Count_number_of_copies, и при необходимости она удаляется */
DROP PROCEDURE IF EXISTS Count_number_of_copies;
GO

/* проверяется, существует ли временная таблица Temp1, и при необходимости она удаляется */
```

```
DROP TABLE IF EXISTS TEMP1;
GO

CREATE PROCEDURE Count_number_of_copies
@b_id_book varchar(20)       /*Объявляем входную переменную*/
@number int  OUTPUT          /*Объявляем выходную переменную*/
AS
/* Следующая конструкция проверяет, существуют ли записи в таблице «Book» с заданным b_id_book*/
IF NOT EXISTS (SELECT * FROM Book WHERE b_id_book = @b_id_book)
RETURN 0       /* Вызывает конец процедуры Count_number_of_copies */
SELECT Copy_of_book.cb_b_id_book
INTO TEMP1   /*Сохраняет выбранные поля во временной таблице Temp1*/
FROM Copy_of_book
WHERE cb_b_id_book = @b_id_book
SELECT @number = COUNT(cb_b_id_book)       /* COUNT подсчитывает количество неповторяющихся записей поля b_id_book */
FROM TEMP1
```

Пример хранимой процедуры на удаление из таблицы «Student». Допустимо, если в таблице «Use_of_libr_student» нет ссылающихся записей.

```
CREATE  PROCEDURE   Delete_student
@num int             /* Объявляем входные переменные */
AS                   /* Проверяем, есть ли ссылающиеся записи, если записей нет, разрешается удаление */
IF NOT EXISTS  (SELECT * FROM Use_of_libr_student
                     WHERE  us_s_number_of_library_ticket =@num)
DELETE               /* Оператор удаления */
FROM Student         /* Имя таблицы, откуда нужно удалить */
WHERE                /* Условие удаления – удаляем строку, для которой значение поля us_s_number_of_library_ticket совпадает с нужным */
us_s_number_of_library_ticket = @num
```

Пример хранимой процедуры на вставку в таблицу «Order_of_book». Разрешена, если в таблицах «Book» и «Lecturer» есть записи, на которые будет ссылаться новая запись.

```
PROCEDURE  Create_new_order
@quantity int,
@order_date datetime,
@number int,
@b_id_book varchar(20)
AS   /* Проверяем, есть ли запись в таблице «Order_of_book» с такими же значениями ключевых полей, как у новой записи */
```

```
IF EXISTS (SELECT * FROM Order_of_book
            WHERE ob_b_id_book = @b_id_book
            AND ob_l_number_of_library_ticket = @number)
RETURN 0          /* Если есть, завершаем выполнение процедуры */
IF EXISTS (SELECT * FROM Lecturer
            WHERE l_number_of_library_ticket = @number)
/*Проверяем, есть ли в «Lecturer» соответствующая запись */
IF EXISTS (SELECT * FROM Book
             WHERE b_id_book = @b_id_book)
/* Проверяем, есть ли в «Book» соответствующая запись */
INSERT INTO  Order_of_book       /* Указываем таблицу, в которую вставляем запись  */
VALUES  (@quantity, @order_date, @number, @b_id_book)    /* Указываем, какие значения  */
```

Пример хранимой процедуры на обновление таблицы «Student» (изменение фамилии студента).

```
CREATE PROCEDURE Update_student
@number int,          /* Объявляем входные переменные */
@lname varchar(20)
AS
IF EXISTS (SELECT * FROM Student      /* Проверяем, существуют ли студенты, */
    WHERE s_number_of_library_ticket = @number)       /* номер читательского билета которых равен искомому */
UPDATE Student          /* Если такие есть, обновляем таблицу «Student» */
SET s_last_name=@lname     /* полю фамилия присваиваем новое значение */
    WHERE s_number_of_library_ticket = @number      /* если номер читательского билета записи равен искомому */
```

Задание

В разрабатываемой БД необходимо реализовать 20 хранимых процедур, в том числе для вставки, удаления, изменения данных. Должны быть использованы стандартные функции SQL (не менее 20 различных функций), а также выходные параметры процедур.

Содержание отчета: тема и цель работы; SQL-код 20 хранимых процедур, SQL-код вызова хранимых процедур на исполнение.

Контрольные вопросы

1 Что такое хранимая процедура? Для чего используется?
2 Какие виды параметров могут использоваться в процедуре?

3 Как производится средствами T-SQL создание, модификация и удаление хранимых процедур? Как создаются хранимые процедуры на вставку, изменение и удаление данных?

4 Как вызвать средствами T-SQL процедуру с входными и выходными параметрами на выполнение?

5 Описать управление процессом компиляции хранимой процедуры.

19 Язык SQL. Работа с триггерами

Цель: научиться создавать триггеры в MS SQL Server Management Studio.

Теоретические положения

Триггер – это специальный тип хранимых процедур, который запускается автоматически на стороне сервера при выполнении тех или иных действий с данными таблицы. Каждый триггер привязывается к конкретной таблице [4, 11].

Напрямую обратиться к триггеру нельзя. Он вызывается автоматически при наступлении соответствующего события в БД – добавление новой строки в таблицу, изменение или удаление строки таблицы. Триггер может срабатывать, когда соответствующее действие с БД выполняет клиентское приложение, хранимая процедура или триггер (другой или тот же самый).

Триггеры DML вызываются при выполнении операторов INSERT, UPDATE или DELETE. Можно указать время вызова триггера:

– AFTER – триггер вызывается после всех действий оператора, если оператор был выполнен успешно. Синонимом в синтаксисе оператора создания триггера является FOR. Также до запуска триггера должны успешно завершиться все каскадные действия и проверки ограничений, на которые есть ссылки;

– INSTEAD OF – триггер вызывается вместо действий, заданных оператором INSERT, UPDATE или DELETE.

Триггеры DDL активируются в ответ на разные события языка описания данных DDL. Эти события прежде всего соответствуют инструкциям Transact-SQL CREATE, ALTER, DROP и некоторым системным хранимым процедурам, которые выполняют схожие с DDL операции.

Триггеры входа могут срабатывать в ответ на событие LOGON, которое возникает при создании пользовательского сеанса.

Для создания триггера используется следующая инструкция T-SQL.

```
CREATE TRIGGER  [ schema_name . ] Trigger_name
ON Table_name | View_name
{ FOR | AFTER | INSTEAD OF }
{ [ INSERT ] [ , ] [ UPDATE ] [ , ] [ DELETE ] }
[ WITH APPEND ]
AS
sql_statement [...n] }
```

Trigger_name – задает имя триггера, с помощью которого он будет распознаваться хранимыми процедурами и командами T-SQL. Имя триггера должно быть уникальным в пределах БД. Оно должно характеризовать действие, выполняемое им, записываться в формате <глагол_объект> и содержать в себе имя таблицы и название момента срабатывания.

Table_name | View_name – имя таблицы БД (или представления), к которой будет привязан триггер.

[DELETE] [,] [INSERT] [,] [UPDATE] – определяет инструкции изменения данных, при применении которых к таблице или представлению срабатывает триггер DML. При создании триггера должно быть указано хотя бы одно из этих ключевых слов, и допускается создание триггера, реагирующего на две или три команды в любом сочетании.

WITH APPEND – указание этого ключевого слова требуется для обеспечения совместимости с более ранними версиями MS SQL Server.

sql_statement – определяет набор команд, которые будут выполняться при запуске триггера.

Далее приведен пример триггера, который будет запрещать удаление записей таблицы «Issuing_books», если текущий пользователь не владелец базы данных и если поле «дата выдачи» содержит какое-либо значение.

```
CREATE TRIGGER Ondelete_issuing_books   /* Объявляем имя триггера */
ON Issuing_books                        /* имя таблицы, с которой связан триггер */
FOR DELETE  /*Указываем операцию, на которую будет срабатывать триггер (здесь на  удаление)*/
AS
IF ( SELECT COUNT(*)        /*проверяет*/
FROM Issuing_books          /*записи из таблицы «Issuing_books»*/
WHERE Issuing_books.ib_date_of_issue IS NOT NULL) > 0   /*условие проверяет наличие записи в поле «ib_date_of_issue». Если COUNT(*) возвращает значение, отличное от нуля (т. е. запись есть), то первое условие IF не выполнено*/
AND (CURRENT_USER <> 'dbo')         /*вызывается функция определения имени текущего пользователя и проверяется, владелец ли он*/
BEGIN
PRINT 'у вас нет прав на удаление этой записи'      /*выдача сообщения о неудаче операции*/
ROLLBACK TRANSACTION    /*откат (отмена) транзакции*/
END
```

Задание

В разрабатываемой БД необходимо реализовать 10 триггеров, реагирующих на различные команды (INSERT, DELETE, UPDATE) и содержащих не менее 10 различных стандартных функций SQL.

Содержание отчета: тема и цель работы; SQL-код 10 триггеров.

Контрольные вопросы

1 Что такое триггер? Какие типы триггеров различают?
2 Как создать триггер?
3 С помощью каких команд T-SQL можно изменить или удалить триггер?

20 Язык SQL. Работа с курсорами

Цель: научиться создавать курсоры в MS SQL Server Management Studio.

Теоретические положения

Курсоры в SQL Server представляют собой механизм обмена данными между сервером и клиентом. Курсор позволяет клиентским приложениям работать не с полным набором данных, а только с одной или несколькими строками.

Существует четыре основных типа курсоров, различающихся по предоставляемым возможностям, – статические, динамические, последовательные и ключевые. Тип курсора определяется на стадии его создания и не может быть изменен. Более подробно курсоры описаны в конспекте лекций и в [4, 11].

Далее рассмотрен пример создания курсора для просмотра информации о студентах и выдачи информации об их числе.

```
DECLARE curs1 CURSOR
GLOBAL          /* Создается глобальный курсор, который
                 будет существовать до закрытия данного соединения*/
SCROLL          /* Создает прокручиваемый курсор */
KEYSET          /* Будет создан ключевой курсор */
TYPE_WARNING
FOR
SELECT          /* Какие поля будут показаны в курсоре */
Student.s_number_of_library_ticket, Student.s_last_name, Student.s_first_name,
Student.s_patronymic, Student.s_year_of_entrance, Student.s_faculty
FROM Student                /* Из какой таблицы выбираются данные */
FOR READ ONLY               /* Только для чтения */
OPEN GLOBAL curs1           /* открываем глобальный курсор */
DECLARE @@Counter int       /* объявляем переменную*/
SET @@Counter =@@CURSOR_ROWS  /*присваиваем ей число рядов
курсора */
Select @@Counter            /* выводим результат на экран */
CLOSE curs1                 /* закрываем курсор */
DEALLOCATE curs1            /* освобождаем курсор */
```

Задание

В разрабатываемой БД необходимо реализовать 5 курсоров статического и динамического типов, содержащих не менее пяти различных функций.

Содержание отчета: тема и цель работы; SQL-код 5 курсоров.

Контрольные вопросы

1 Что такое курсор? Какие типы курсоров различают?

2 Что такое полный и результирующий наборы строк?

3 Какие основные операции выделяют при работе с курсором? С помощью каких команд T-SQL реализуются основные операции?

21 Назначение прав доступа пользователям к объектам базы данных средствами T-SQL

Цель: изучить основы управления правами доступа к объектам базы данных в СУБД MS SQL Server.

Теоретические положения

Вопросы управления правами доступа к объектам БД подробно описаны в [1, 6, 7, 10, 11]. Здесь же рассмотрен SQL-код основных инструкций.

С каждым защищаемым объектом в SQL Server связаны разрешения, которые могут быть предоставлены участнику. Управление разрешениями в компоненте Database Engine осуществляется на уровне сервера (назначение разрешений именам входа и ролям сервера) и на уровне базы данных (назначение разрешений пользователям и ролям базы данных).

Для предоставления разрешений permission на защищаемый объект securable участнику principal используется инструкция GRANT, общая концепция которой имеет следующий вид: GRANT <some permission> ON <some object> TO <some user, login, or group>.

Синтаксис инструкции GRANT:

```
GRANT { ALL [ PRIVILEGES ] }   | permission [ ( column [ ,...n ] ) ] [ ,...n ]
    [ ON [ class :: ] securable ] TO principal [ ,...n ]
    [ WITH GRANT OPTION ] [ AS principal ]
```

Инструкция DENY запрещает разрешение для участника. Предотвращает наследование разрешения участником через его членство в группе или роли. DENY имеет приоритет над всеми разрешениями, но не применяется к владельцам объек-

тов или членам с предопределенной ролью сервера sysadmin (членам предопределенной роли сервера sysadmin и владельцам объектов не может быть отказано в разрешениях). Синтаксис инструкции DENY:

```
DENY   { ALL [ PRIVILEGES ] } | <permission>  [ ( column [ ,...n ] ) ] [ ,...n ]
   [ ON [ <class> :: ] securable ]
   TO principal [ ,...n ]
   [ CASCADE] [ AS principal ]  [;]
```

Инструкция REVOKE удаляет разрешение, выданное или запрещенное ранее (иногда говорят, что REVOKE используется для неявного отклонения доступа к объектам БД):

```
REVOKE [ GRANT OPTION FOR ]
    {   [ ALL [ PRIVILEGES ] ]   |   permission [ ( column [ ,...n ] ) ] [ ,...n ]  }
    [ ON [ class :: ] securable ]
    { TO | FROM } principal [ ,...n ]
    [ CASCADE] [ AS principal ]
```

Охарактеризуем назначение параметров инструкций.

ALL – этот параметр устарел и сохранен только для поддержки обратной совместимости. Он не предоставляет все возможные разрешения (см. документацию https://docs.microsoft.com/). Вместо ALL следует предоставлять конкретные разрешения.

PRIVILEGES – включено для обеспечения совместимости с требованиями ISO. Не изменяет работу ALL.

permission – имя разрешения, которое предоставляется пользователю. Можно предоставлять одновременно несколько разрешений.

column – указывает имя столбца таблицы, на который предоставляется разрешение.

class – указывает класс защищаемого объекта, для которого предоставляется разрешение. Квалификатор области :: является обязательным.

securable – указывает защищаемый объект, на который предоставляется разрешение.

TO principal – имя участника. Состав участников, которым можно предоставлять разрешения, меняется в зависимости от защищаемого объекта.

WITH GRANT OPTION – позволяет пользователю, получающему разрешение, получить также возможность предоставлять данное разрешение другим участникам.

AS principal – указывает, что участник, записанный как предоставляющий разрешение, должен быть участником, отличным от пользователя, выполняющего инструкцию. Предположим, что пользователь Мария – это участник 12, пользователь Павел – участник 15. Мария выполняет GRANT SELECT ON OBJECT::X TO Steven WITH GRANT OPTION AS Paul;. В таблице

sys.database_permissions для параметра grantor_principal_id будет указано значение 15 (Павел), хотя инструкция была выполнена пользователем 12 (Мария). Использовать предложение AS обычно не рекомендуется, за исключением необходимости явного определения цепочки разрешения.

GRANT OPTION FOR – указывает, что возможность предоставлять указанное разрешение будет отменена. Данный аргумент необходим при использовании аргумента CASCADE. Если участник обладает указанным разрешением без параметра GRANT, будет отменено само разрешение.

TO | FROM principal – имя участника. Участники, у которых может быть отменено разрешение на доступ к защищаемому объекту, различны.

CASCADE – позволяет отзывать права не только у данного пользователя, но и у всех пользователей, которым он предоставил данные права. Аргумент CASCADE необходимо использовать совместно с GRANT OPTION FOR. Каскадная отмена разрешения, предоставленного с помощью WITH GRANT OPTION, приведет к отмене разрешений GRANT и DENY для этого разрешения.

Предоставление разрешения удаляет DENY или REVOKE для этого разрешения на данный защищаемый объект. Если то же разрешение запрещено для более высокой области действия, которая содержит данный защищаемый объект, то DENY имеет более высокий приоритет. Однако отмена разрешения на более высоком уровне не имеет более высокого приоритета.

Задание

Для разрабатываемой БД необходимо реализовать систему безопасности: создать одну роль и трех пользователей. В созданную роль нужно включить двух пользователей. Третьего пользователя следует включить в одну из стандартных ролей SQL Server.

Далее необходимо назначить права на все ранее разработанные объекты БД. Следует обратить внимание, что не должно быть объектов базы данных, на которые ни у кого из пользователей прав нет. В отчете указывается, с какими объектами базы данных позволяет пользователям работать каждая роль, а также приводятся фрагменты инструкций SQL, позволяющих создать роли, пользователей и предоставить этим пользователям права доступа к объектам базы данных.

Назначение прав доступа на объекты базы данных (таблицы, представления, хранимые процедуры) должно быть проиллюстрировано таблицей 21.1 с помощью общепринятых сокращений, произошедших от аббревиатуры CRUD – Create, Read, Update, Delete (C – право на создание записи, R – право на чтение записи, U – право на обновление записи, D – право на удаление записи).

Таблица 21.1 – Перечень прав пользователей (категорий пользователей) относительно объектов базы данных

Имя объекта базы данных	Пользователь 1	Пользователь 2	Пользователь *N*	Приложение	Гость	Администратор
Table1	CRU	R	U	R	–	CRUD
View1	R	R	CRU	R	R	CRUD

Содержание отчета: тема и цель работы; описание распределения прав доступа пользователей; SQL-код выполнения задания.

Контрольные вопросы

1 На какие категории можно разделить права в SQL Server?

2 Перечислить стандартные роли сервера и базы данных.

3 Какие команды T-SQL используются для предоставления прав доступа, запрещения и неявного отклонения доступа?

Список литературы

1 **Агальцов, В. П.** Базы данных [Электронный ресурс]: в 2 т. Т. 2: Распределенные и удаленные базы данных : учебник / В. П. Агальцов. – Москва: ФОРУМ; ИНФРА-М, 2018. – 271 с. – Режим доступа: https://znanium.com/catalog/product/1068927]. – Дата доступа: 25.11.2021.

2 **Агальцов, В. П.** Базы данных [Электронный ресурс]: учебник: в 2 кн. Кн. 1: Локальные базы данных / В. П. Агальцов. – Москва: ФОРУМ; ИНФРА-М, 2020. – 352 с.: ил. – Режим доступа: https://znanium.com/catalog/product/1068927. – Дата доступа: 25.11.2021.

3 **Бен-Ган, И.** Microsoft SQL Server 2012. Создание запросов: учебный курс Microsoft: пер. с англ. / И. Бен-Ган, Д. Сарка, Р. Талмейдж. – Москва : Русская редакция, 2015. – 720 с. : ил.

4 **Бондарь, А. Г.** Microsoft SQL Server 2014 / А. Г. Бондарь. – Санкт-Петербург: БХВ-Петербург, 2015. – 592 с.: ил.

5 **Голицына, О. Л.** Базы данных [Электронный ресурс]: учебное пособие / О. Л. Голицына, Н. В. Максимов, И. И. Попов. – 4-е изд., перераб. и доп. – Москва: ФОРУМ; ИНФРА-М, 2020. – 400 с. – Режим доступа: https://znanium.com/catalog/product/1053934. – Дата доступа: 25.11.2021.

6 **Дадян, Э. Г.** Данные: хранение и обработка [Электронный ресурс]: учебник / Э. Г. Дадян. – Москва: ИНФРА-М, 2020. – 205 с. – Режим доступа: https://znanium.com/catalog/product/1045133. – Дата доступа: 25.11.2021.

7 **Кузин, А. В.** Базы данных: учебное пособие для студентов высших учебных заведений / А. В. Кузин, С. В. Левонисова. – 6-е изд., стер. – Москва: Академия, 2016. – 320 с.

8 **Куликов, С. С.** Реляционные базы данных в примерах: практическое пособие для программистов и тестировщиков [Электронный ресурс] / С. С. Куликов. – Минск: Четыре четверти, 2020. – 424 с. – Режим доступа: http://svyatoslav.biz/relational_databases_book/. – Дата доступа: 25.11.2021.

9 **Куликов, С. С.** Работа с MySQL, MS SQL Server и Oracle в примерах [Электронный ресурс]: практическое пособие / С. С. Куликов. – Минск: БОФФ, 2016. – 556 с. – Режим доступа: http://svyatoslav.biz/database_book/. – Дата доступа: 28.11.2021.

10 **Полищук, Ю. В.** Базы данных и их безопасность [Электронный ресурс]: учебное пособие / Ю. В. Полищук, А. С. Боровский. – Москва: ИНФРА-М, 2020. – 210 с. – Режим доступа: https://znanium.com/catalog/product/1011088. – Дата доступа: 25.11.2021.

11 **Шустова, Л. И.** Базы данных [Электронный ресурс]: учебник / Л. И. Шустова, О. В. Тараканов. – Москва: ИНФРА-М, 2017. – 304 с. – Режим доступа: www.dx.doi.org/10.12737/11549. – Дата доступа: 28.11.2021.