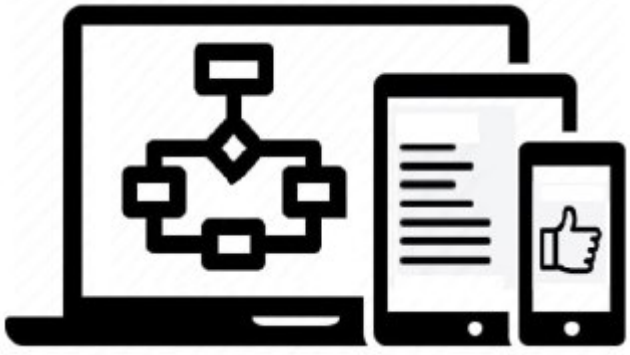
МЕЖГОСУДАРСТВЕННОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ОБРАЗОВАНИЯ
«БЕЛОРУССКО-РОССИЙСКИЙ УНИВЕРСИТЕТ»

Кафедра «Программное обеспечение информационных технологий»

СИСТЕМЫ УПРАВЛЕНИЯ БАЗАМИ ДАННЫХ

Методические рекомендации к лабораторным работам для студентов специальности 1-53 01 02 «Автоматизированные системы обработки информации» дневной и заочной форм обучения

Часть 1



Могилев 2023

УДК 004.65
ББК 32.973.26-0.18.2
С89

Рекомендовано к изданию
учебно-методическим отделом
Белорусско-Российского университета

Одобрено кафедрой «Программное обеспечение информационных технологий» «31» мая 2023 г., протокол № 11

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

Рецензент Ю. С. Романович

Методические рекомендации содержат описание пяти лабораторных работ», выполняемых в первом семестре изучения дисциплины «Системы управления базами данных». Рассматриваются основы работы с Microsoft SQL Server и языком Transact-SQL.

Учебное издание

СИСТЕМЫ УПРАВЛЕНИЯ БАЗАМИ ДАННЫХ

Часть 1

Ответственный за выпуск	В. В. Кутузов
Корректор	Т. А. Рыжикова
Компьютерная верстка	Е. В. Ковалевская

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

Введение

Целью учебной дисциплины «Системы управления базами данных» (СУБД) является формирование профессиональных компетенций для работы с современными системами управления базами данных, основами эксплуатации баз данных (БД) в составе автоматизированных систем обработки информации, внедряемых в различных областях науки, техники и экономики.

В результате освоения учебной дисциплины студент:

а) ознакомится с:

– языком SQL;

– способами работы с реляционными базами данных;

– способами реализации распределенной и параллельной обработки данных;

– возможностями современных СУБД для построения систем поддержки принятия решений;

б) научится:

– практически создавать и администрировать базы данных;

– создавать интерфейс с базами данных;

– организовывать работу в многопользовательской базе данных;

– устанавливать и конфигурировать серверные и клиентские приложения баз данных;

в) овладеет:

– методами, средствами и технологиями разработки информационных моделей и их программной реализации в выбранной СУБД;

– методами программирования систем на основе баз данных.

Методические рекомендации содержат описание пяти лабораторных работ», выполняемых при изучении дисциплины «Системы управления базами данных», задания, контрольные вопросы, список литературы [1–11], которую необходимо использовать при подготовке к защите лабораторных работ.

1 Технология создания баз данных на основе промышленной СУБД

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) [3]. 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 [3–5, 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 сведения 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?

2 Создание sql-скрипта заполнения базы данных

Цель: получить навыки генерации sql-скриптов создания и 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 (кнопка Advanced): 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].

3 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 БД, 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 } ]
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>.

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 вычисляемых столбцов. 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 кеширующие (материализованные, индексированные) представления?

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

4 Перечислить ограничения при создании представлений.

4 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 значе-
ние 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 встав-
ляем 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.

5 Язык SQL. Создание пользовательских функций

Цель: научиться создавать пользовательские функции с использованием языка T-SQL.

Теоретические положения

Пользовательские функции (*User Defined Functions* (*UDF*)) – это процедуры T-SQL, которые инкапсулируют повторно используемый код T-SQL, могут принимать параметры и возвращать либо скалярные значения, либо таблицы [4].

В отличие от хранимых процедур, пользовательские функции встроены в инструкции T-SQL, исполняются как часть команды T-SQL и не могут выполняться с помощью команды EXECUTE. Пользовательские функции имеют доступ к данным SQL Server, но не могут выполнять инструкции DDL или изменять любые данные в постоянных таблицах с помощью инструкций DML.

Функции, определяемые пользователем, могут быть скалярными или табличными. Скалярная функция возвращает скалярное значение (число). Это означает, что в предложении RETURNS скалярной функции задается один из стандартных типов данных. Функции являются табличными, если предложение RETURNS возвращает набор строк.

Виды пользовательских функций:

– *скалярная функция* – возвращает вызывающей стороне одно значение;

– *функция с табличным значением* – возвращает таблицу и может появляться в предложении FROM запроса T-SQL. Функция с табличным значением, состоящая из одной строки кода, называется *встроенной пользовательской функцией с табличным значением*. Функция с табличным значением, состоящая из нескольких строк кода, называется *многооператорной возвращающей табличное значение пользовательской функцией*.

Пользовательские скалярные функции могут появляться в любом месте запроса, где может появляться выражение, возвращающее одно значение (например, в списке столбца SELECT). Весь код внутри пользовательской скалярной функции должен быть заключен в блок BEGIN/END.

В SSMS, если щелкнуть правой кнопкой мыши в окне запроса, выбрать команду Insert Snippet (Вставить фрагмент) и вставить фрагмент для скалярной функции, то можно увидеть следующие выходные данные:

```
CREATE FUNCTION [dbo].[FunctionName]
(  @param1 int,
   @param2 int  )
RETURNS INT
AS
BEGIN
   RETURN @param1 + @param2
END
```

На основе данного фрагмента можно, например, создать простую скалярную функцию, которая возвращает количество наименований книг определенного издательства.

```
IF OBJECT_ID('fn_book_count_by_publisher', 'FN') IS NOT NULL
DROP FUNCTION fn_book_count_by_publisher
GO

-- Transact-SQL Scalar Function
CREATE FUNCTION fn_book_count_by_publisher
(@publisher_name VARCHAR(50))
RETURNS INT
AS
BEGIN
   DECLARE @book_count INT
```

```
    SELECT @book_count = COUNT(*)
    FROM Book
      WHERE b_publisher = @publisher_name
    RETURN @book_count
END;
GO
```

Далее приведены примеры вызова этой функции внутри запроса T-SQL:

```
SELECT dbo.fn_book_count_by_publisher('ИНФРА-М');

SELECT DISTINCT [dbo].[Book].b_publisher,
dbo.fn_book_count_by_publisher('ИНФРА-М')
FROM Book
  WHERE [b_publisher] LIKE ('ИНФРА-М');
```

Встроенная пользовательская функция с табличным значением (inline table-valued function) содержит одну инструкцию SELECT, которая возвращает таблицу. Рассмотрим пример Inline table-valued function для таблицы Book, который будет возвращать книги, у которых название начинается с заданной подстроки.

```
-- Transact-SQL Inline Table-Valued Function
CREATE FUNCTION dbo.FilterBooksBySubstring
(@substring varchar(50))
RETURNS TABLE
AS
RETURN
(
    SELECT b_id_book, b_title, b_publisher
    FROM Book
    WHERE b_title LIKE @substring + '%'
);
GO
```

Пример вызова данной функции:

```
SELECT *
FROM dbo.FilterBooksBySubstring('Война и мир');
```

Таким образом, inline table-valued function содержит только один оператор SELECT в теле функции, который определяет структуру результирующей таблицы. Такая функция может быть использована в запросах как обычная таблица. Как правило, она применяется для простых запросов, возвращающих небольшое количество строк.

Поскольку встроенная функция с табличным значением не выполняет никаких других операций, оптимизатор обрабатывает ее точно так же, как представление. Можно даже применять к ней инструкции INSERT, UPDATE и DELETE, как для представления. Однако следует учитывать, что изменения, внесенные в результирующую таблицу через функцию, не сохраняются в исходной таблице, т. к. функция не изменяет данные в базе данных напрямую.

Рассмотрим пример *многооператорной возвращающей табличное значение пользовательской функцией* (Transact-SQL Multi-Statement Table-Valued Function) для таблицы Book, который будет возвращать книги, у которых длина названия больше заданной.

```
-- Transact-SQL Multi-Statement Table-Valued Function
CREATE FUNCTION dbo.FilterBooksByTitleLength (@length int)
RETURNS @BooksTable TABLE
  ( b_id_book int,
    b_title varchar(100),
    b_publisher varchar(50)
)
AS
BEGIN
    DECLARE @TempTable TABLE (b_id_book int, b_title varchar(100),
b_publisher varchar(50));

    INSERT INTO @TempTable (b_id_book, b_title, b_publisher)
    SELECT b_id_book, b_title, b_publisher
    FROM Book;

    DELETE FROM @TempTable
    WHERE LEN(b_title) <= @length;

    INSERT INTO @BooksTable (b_id_book, b_title, b_publisher)
    SELECT b_id_book, b_title, b_publisher
    FROM @TempTable;

    RETURN;
END;
```

Пример вызова функции:

```
SELECT *
FROM dbo.FilterBooksByTitleLength(10);
```

Таким образом, multi-statement table-valued function может содержать один или несколько операторов SQL, например, SELECT, INSERT, UPDATE, DELETE и т. д. Структуру результирующей таблицы можно определить в теле функции.

В теле функции можно изменять и агрегировать результирующую таблицу. Синтаксис функции включает оператор BEGIN и END. Функция данного типа используется для объединения с другими таблицами в запросах и для генерации более сложных результатов, чем inline table-valued function, которая содержит только один оператор SELECT.

В пользовательских функциях могут быть использованы следующие недетерминированные встроенные функции: CURRENT_TIMESTAMP(), GETDATE(), GETUTCDATE(). Нельзя использовать следующие недетерминированные встроенные функции: NEWID(), RAND().

Пользовательские функции нельзя использовать для выполнения действий, изменяющих состояние базы данных.

Пользовательские функции не могут возвращать несколько результирующих наборов (в данном случае используются хранимые процедуры).

Обработка ошибок в пользовательских функциях ограниченна, т. к. не поддерживаются TRY...CATCH, @ERROR или RAISERROR.

Пользовательские функции не могут использовать динамические таблицы SQL или временные таблицы. Табличные переменные разрешены к использованию. Использование операторов SET не допускается.

Задание

Необходимо реализовать 10 различных пользовательских функций.

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

Контрольные вопросы

1 Что такое пользовательская функция? Для чего используется?
2 Как создать и вызвать пользовательские функции различных типов?
3 Каковы ограничения при создании пользовательских функций?

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, 2021. – 271 с. – Режим доступа: https://znanium.com/catalog/document?id=377105]. – Дата доступа: 25.04.2023.

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/document?id=398558. – Дата доступа: 25.04.2023.

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 **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.

6 **Куликов, С. С.** Реляционные базы данных в примерах: практическое пособие для программистов и тестировщиков [Электронный ресурс] / С. С. Куликов. – Минск: Четыре четверти, 2023. – 424 с. – Режим доступа: https://svyatoslav.biz/relational_databases_book/. – Дата доступа: 25.04.2023.

7 **Куликов, С. С.** Работа с MySQL, MS SQL Server и Oracle в примерах [Электронный ресурс]: практическое пособие / С. С. Куликов. – Минск: БОФФ, 2022. – 592 с. – Режим доступа: http://svyatoslav.biz/database_book/. – Дата доступа: 25.04.2023.

8 **Мартишин, С. А.** Базы данных. Практическое применение СУБД SQL и NoSQL-типа для проектирования информационных систем [Электронный ресурс]: учебное пособие / С. А. Мартишин, В. Л. Симонов, М. В. Храпченко. – Москва: ФОРУМ ; ИНФРА-М, 2022. – 368 с. – Режим доступа: https://znanium.com/catalog/ product/1873270. – Дата доступа: 25.04.2023.

9 **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: 25.04.2023.

10 **Тарасов, С. В.** СУБД для программиста: базы данных изнутри [Электронный ресурс] / С. В. Тарасов. – Москва: СОЛОН-Пресс, 2020. – 320 с. – Режим доступа: https://znanium.com/catalog/product/1227737. – Дата доступа: 25.04.2023.

11 **Шустова, Л. И.** Базы данных [Электронный ресурс]: учебник / Л. И. Шустова, О. В. Тараканов. – Москва: ИНФРА-М, 2017. – 304 с. – Режим доступа: https://znanium.com/catalog/document?id=13826. – Дата доступа: 25.04.2023.