Межгосударственное образовательное учреждение высшего образования
«БЕЛОРУССКО-РОССИЙСКИЙ УНИВЕРСИТЕТ»

Кафедра «Автоматизированные системы управления»

БАЗЫ И БАНКИ ДАННЫХ

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

Могилев 2018

СОДЕРЖАНИЕ

Лабораторная работа № 17. Создание отношений между таблицами средствами SQL

2 часа

Цель: создание отношений между таблицами средствами T-SQL.

17.1 Теоретические положения

Вопросы создания отношений между таблицами описаны в конспекте лекций и в [2, 3, 6, 15].

Для реализации 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) таблицей, а таблица, содержащая соответствующий первичный ключ, называется ссылочной или родительской таблицей.

Например, создадим две таблицы, связанные отношением 1:1. Столбец ссылки id_a_link таблицы TableB нужно сделать уникальным внешним ключом. Это гарантирует, что в таблице TableB может быть только одна запись, которая соответствует значению в столбце PRIMARY KEY в таблице TableA.

```
CREATE  TABLE   TableA (
id_a   INT   PRIMARY KEY   IDENTITY(1,1),
name   VARCHAR(255));

CREATE  TABLE   TableB (
id_b   INT   PRIMARY KEY   IDENTITY(1,1),
name   VARCHAR(255),
```

```
id_a_link   INT   UNIQUE,
FOREIGN KEY  (id_a_link)   REFERENCES   TableA  (id_a));
```

Задание

Необходимо установить отношения между резервными таблицами из лабораторной работы № 16, соответствующие отношениям между основными таблицами, средствами T-SQL.

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

Контрольные вопросы

1 Какие типы отношений поддерживаются в MS SQL Server?
2 Для чего предназначены предложения FOREIGN KEY и REFERENCES?
3 Какие ограничения на значения столбцов можно накладывать?

Лабораторная работа № 18. Язык SQL. Добавление, изменение и удаление данных в таблицах средствами SQL

2 часа

1 Добавление данных в таблицу средствами T-SQL

1.1 Использование команды INSERT

Для вставки данных в таблицу средствами T-SQL используется команда INSERT, имеющая следующий синтаксис [4]**:**

```
INSERT [INTO] {
имя_таблицы | имя_ представления
| OPENQUERY  | OPENROWSET
} {
[ (список_столбцов) ]
{
VALUES
( { DEFAULT | NULL
| выражение }[,... n ] ) | временная_таблица | инструкция_выполнения
}
}
|  DEFAULT VALUES
```

Ключевое слово INSERT и необязательное ключевое слово INTO вводят инструкцию. Ключевое слово INTO используется для удобочитаемости.

Аргумент «имя_ таблицы» задает целевую таблицу, в которую необходимо вставить данные. Можно в качестве цели вставки задать аргумент «имя_представления» или результаты выполнения функций OPENQUERY и OPENROWSET.

Аргумент «список_столбцов» представляет необязательный 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.

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.

1.2 Ограничения команды INSERT

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 в явном виде не указан список столбцов таблицы, то инструкция INSERT будет пытаться вставить значения в каждый столбец таблицы в порядке предоставления значений. Со списком столбцов или без него, инструкция INSERT работает только в том случае, если SQL Server может определить, какое значение вставлять в каждый столбец таблицы. Это означает, что для каждого столбца верно одно из следующих утверждений [4]:

- инструкция INSERT задает некоторое значение;
- столбец является столбцом идентификатора;
- столбец имеет значение по умолчанию;
- столбец имеет тип данных timestamp;
- столбец может содержать пустые значения.

Если необходимо вставить отдельное значение в столбец идентификатора, то параметру таблицы IDENTITY_INSERT следует присвоить значение ON, а само это значение должно быть описано явным образом в инструкции INSERT [4].

Если введено ключевое слово DEFAULT для столбца, который не имеет значения по умолчанию, то инструкция вставит значение NULL (разумеется, если в столбце разрешены пустые значения). В противном случае инструкция INSERT вызовет ошибку, и данные вставлены не будут [4].

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

1.3 Примеры команды INSERT

Самая простая ситуация использования инструкции INSERT сводится к работе с таблицей, которая имеет значения по умолчанию для всех столбцов. Для создания такой таблицы используется следующий синтаксис:

```
CREATE TABLE Продукт.Количество (
PreferenceID int IDENTITY (1, 1) NOT NULL,
Name nvarchar(50) NULL,
Value int)
```

Эта таблица содержит один столбец со свойством IDENTITY, а все остальные столбцы могут содержать значения NULL. Таким образом, можно вставить строку в эту таблицу, используя директиву DEFAULT VALUES:

```
INSERT INTO Продукт.Количество DEFAULT VALUES
```

При желании можно задать значения для набора столбцов во время вставки:

```
INSERT INTO Продукт.Количество (Name, Value)
VALUES ('Сахар',  10)
```

Не обязательно перечислять столбцы в том порядке, в котором они выводятся в таблице. Главное, чтобы списки столбцов и значений соответствовали друг другу.

При вставке значений в таблицу со столбцом идентификаторов можно задать значение для этого столбца обычным способом. Однако, используя директиву SET IDENTITY_INSERT, можно задать значение для столбца идентификаторов следующим образом:

```
SET IDENTITY_INSERT Продукт.Количество ON INSERT INTO Продукт.Количество (ПродуктID, Name, Value) VALUES  (17285,  'Сахар', 10')
```

Если осуществляется вставка значений во все столбцы, то можете опустить в инструкции список столбцов. Однако при этом инструкция станет более запутанной, поэтому рекомендуется всегда включать список столбцов в явном виде. Отметим, что если таблица содержит столбец IDENTITY, список столбцов обязательно должен быть определен в явном виде.

1.4 Синтаксис инструкции SELECT INTO

Инструкция SELECT INTO является вариацией простой инструкции SELECT. Схематически ее синтаксис выглядит так [4]:

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

Инструкция SELECT INTO в целом идентична инструкции SELECT.

Новым ключевым 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 соот-

ветствующих столбцов в списке SELECT. Другими словами, инструкция SELECT INTO использует результаты выполнения инструкции SELECT и преобразует их в постоянную таблицу [2].

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 их 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.

2 Изменение данных в таблицах средствами T-SQL

2.1 Использование команды UPDATE

Данные, хранящиеся в таблицах баз данных, как правило, не являются статичными. 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 чтобы она вычисляла изменения в столбце во всех строках таблицы. Можно также написать инструкцию, которая будет изменять все строки из множества столбцов [3].

Синтаксис инструкции UPDATE [2]:

```
UPDATE {
имя_таблицы | имя_представления
| OPENQUERY | OPENROWSET
}
SET {
имя_столбца = {выражение | DEFAULT | NULL}
| @ переменная = выражение
| @переменная-столбец = выражение
{
[FROM {исходная_таблица} [,... n]]
[WHERE условие_поиска]
}
[OPTION (подсказка_запроса [,...n]}]
```

Ниже приведен анализ синтаксиса инструкции UPDATE [2].

– Ключевое слово UPDATE идентифицирует инструкцию.

– В качестве исходного материала для обновляемых строк нужно задать либо имя таблицы или представления, либо результаты выполнения функций OPENQUERY или OPENROWSET.

– 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.

Если инструкция UPDATE используется для обновления с помощью представления, то такое представление должно быть обновляемым. Кроме того, инструкция UPDATE может повлиять только на одну таблицу в представлении.

Можно использовать инструкцию UPDATE для обновления представления в столбце идентификатора. Если требуется обновить столбец идентификатора, следует использовать инструкцию DELETE для удаления текущей строки и инструкцию INSERT для вставки измененных данных в качестве новой строки.

В инструкции UPDATE можно использовать только выражения, возвращающие одно значение.

Если инструкция UPDATE нарушает некоторое ограничение таблицы независимо от того, является ли оно реальным ограничением, правилом использования пустых значений или настройкой типа данных, то выполнение инструкции UPDATE будут отменено, и программа выдаст сообщение об ошибке. Если инструкция UPDATE обновляет множество строк, то в случае нарушения ограничения хотя бы одной из них никаких изменений в таблицу внесено не будет.

Если некоторое выражение в инструкции UPDATE генерирует арифметическую ошибку (например, деление на нуль), то обновление не будет выполнено, а программа выдаст сообщение об ошибке. Кроме того, такие ошибки отменяют выполнение остальной части пакета, содержащего инструкцию UPDATE.

Если инструкция UPDATE, содержащая столбцы и переменные, обновляет множество строк, то переменные будут содержать значения только одной из обновляемых строк (причем не известно заранее, какая именно строка будет источником этого значения).

Если инструкция UPDATE влияет на строки в таблице, в которой есть триггер INSTEAD OF UPDATE, то вместо инструкции UPDATE будут выполняться инструкции в триггере [4].

2.2 Примеры инструкции UPDATE

Самая простая инструкция UPDATE выполняет одно изменение, которое влияет на все строки таблицы. Например, можно изменить стоимость всех продуктов, перечисленных в таблице Продукция, на 20,00 долл. с помощью следующей инструкции:

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

Можно изменить для всех покупателей рейтинга на значение, равное 200:

```
UPDATE Customers
SET rating = 200;
```

Замена значения столбца во всех строках таблицы, как правило, используется редко. Поэтому в команде UPDATE можно использовать предикат. Для выполнения указанной замены значений столбца rating, для всех покупателей, которые обслуживаются продавцом Giovanni (snum = 1003), следует ввести:

```
UPDATE Customers SET rating = 200
WHERE snum = 1001;
```

В предложении SET можно указать любое количество значений для столбцов, разделенных запятыми:

```
UPDATE emp SET job = 'MANAGER', sal = sal + 1000, deptno = 20
WHERE ename = 'JONES';
```

В предложении SET можно указать значение NULL без использования какого-либо специального синтаксиса (например, такого как IS NULL). Таким образом, если нужно установить все рейтинги покупателей из Лондона (city = 'London') равными NULL-значению, необходимо ввести:

```
UPDATE Customers SET rating = NULL WHERE city = 'London';
```

Если столбец может содержать пустые значения и не имеет явного значения по умолчанию, при использовании ключевого слова DEFAULT ему будет присвоено значение NULL. Если столбец не может содержать пустых значений и не имеет явного значения по умолчанию, то при использовании умолчаний SQL Server выдаст ошибку.

Чтобы явным образом присвоить пустое значение столбцу, который может такие значения содержать, используется ключевое слово NULL:

```
UPDATE Продукция SET Size = NULL
```

Примечание. Несмотря на то, что эта инструкция выглядит, как конструкция сравнения с пустым значением, на нее не будет влиять параметр ANSI NULLS, поскольку здесь оператор равенства используется для присвоения, а не для сравнения [4].

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

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

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

3 Удаление данных из таблиц средствами T-SQL

3.1 Использование команды DELETE

Для удаления записей при помощи запросов из существующей таблицы можно использовать инструкцию DELETE. Инструкция гибкая и позволяет точно указывать записи, которые нужно удалить.

Инструкция DELETE имеет множество опций, но базовый синтаксис ее довольно прост [4]:

```
DELETE [FROM]
{
имя_таблицы | имя_представления
| OPENQUERY | OPENROWSET | OPENDATASOURCE
}
[FROM исходная_таблица] [WHERE условия_поиска]
[OPTION параметры_запроса]
```

Ключевое слово DELETE идентифицирует инструкцию. Необязательное ключевое слово FROM можно использовать для удобочитаемости инструкции

SQL. В качестве исходного объекта для удаляемых строк вам следует задать либо имя таблицы или представления, либо результаты выполнения функции OPENQUERY, OPENROWSET или OPENDATASOURCE. Предложение FROM имеет такой же синтаксис и параметры, что и предложение FROM в инструкции SELECT. Предложение WHERE имеет такой же синтаксис и параметры, как и предложение WHERE в инструкции SELECT. Предложение OPTION можно использовать для задания дополнительных параметров команды [2].

3.2 Ограничения инструкции DELETE

Если инструкция DELETE в качестве исходного материала для удаления строк использует представление, а не таблицу, то это представление должно быть обновляемо.

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].

Если не включить предложение WHERE в инструкцию DELETE, то инструкция удалит все строки из целевой таблицы. Если включить предложение WHERE, то инструкция удалит только те строки, которые оно отберет [4].

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].

3.3 Примеры инструкции DELETE

Простейшая инструкция DELETE удаляет все строки из целевой таблицы. Во избежание нежелательных последствий используется инструкция SELECT INTO для создания копии таблицы Клиенты и выполняется работа уже с этой копией:

```
SELECT * INTO КлиентыCopy FROM Клиенты DELETE 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.

Если нужно сделать инструкцию SQL удобочитаемой, в нее можно вставить ключевое слово FROM.

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

Чтобы удалить отдельную строку, следует включить в инструкцию директиву WHERE, ее отбирающую:

```
SELECT * INTO КлиентыCopy FROM Клиенты
DELETE КлиентыCopy
WHERE Клиенты.ID = 1
DROP TABLE КлиентыCopy
```

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
```

Чтобы убедиться в том, что инструкция DELETE удалит строки, которые назначены для удаления, можно с помощью Enterprise Manager проанализировать результаты выполнения соответствующей инструкции SELECT (в предыдущем примере SELECT * FROM Клиенты WHERE Фамилия LIKE 'A%').

Можно удалить все строки из таблицы:

```
DELETE FROM Клиенты;
```

В данном примере команда DELETE удаляет все строки без исключения.

Можно удалить из таблицы всех клиентов, которые сделали заказ меньше, чем на 100 у.е. :

```
DELETE FROM  Клиенты WHERE Сумма < 100;
```

В данном примере команда DELETE удаляет все строки, которые попадают под условие.

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.

Ниже рассматривается задача удаления всех клиентов, которые не сделали заказы. Можно решить эту задачу путем использования объединения LEFT JOIN и размещения условия для таблицы Orders:

```
SELECT * INTO КлиентыCopy FROM Клиенты DELETE КлиентыCopy
FROM КлиентыCopy LEFT JOIN Заказы ON КлиентыCopy.КонтактID =
Заказ.КонтактID WHERE Заказ.КонтактID IS NULL
DROP TABLE КлиентыCopy
```

Эта инструкция работает, поскольку операция объединения LEFT JOIN создает строки для каждого клиента и контакта и заполняет столбцы из таблицы «Заказ» со значениями NULL для всех контактов без информации в связанной таблице.

3.4 Инструкция TRUNCATE TABLE

Инструкция TRUNCATE TABLE удаляет все строки в целевой таблице, не записывая в журнал транзакций удаление отдельных строк. Инструкция TRUNCATE TABLE похожа на инструкцию DELETE без предложения WHERE, однако TRUNCATE TABLE выполняется быстрее и требует меньших ресурсов системы и журналов транзакций.

Синтаксис:

TRUNCATE TABLE ИмяЦелевойТаблицы;

По сравнению с инструкцией DELETE, инструкция TRUNCATE TABLE обладает следующими преимуществами.

1) Используется меньший объем журнала транзакций. Инструкция DELETE производит удаление по одной строке и заносит в журнал транзакций запись для каждой удаляемой строки. Инструкция TRUNCATE TABLE удаляет данные, освобождая страницы данных, используемые для хранения данных таблиц, и в журнал транзакций записывает только данные об освобождении страниц.

2) Обычно используется меньшее количество блокировок. Если инструкция DELETE выполняется с блокировкой строк, для удаления блокируется каждая строка таблицы. Инструкция TRUNCATE TABLE всегда блокирует таблицу и страницу, но не каждую строку.

3) В таблице остается нулевое количество страниц, без исключений. После выполнения инструкции DELETE в таблице могут все еще оставаться пустые страницы. Например, чтобы освободить пустые страницы в куче, необходима, как минимум, монопольная блокировка таблицы. Если операция удаления не использует блокировку таблицы, таблица будет содержать множество пустых страниц. В индексах после операции удаления могут оказаться пустые страницы, хотя эти страницы будут быстро освобождены процессом фоновой очистки.

Инструкция TRUNCATE TABLE удаляет все строки таблицы, но структура таблицы и ее столбцы, ограничения, индексы и т. п. сохраняются. Чтобы удалить не только данные таблицы, но и ее определение, следует использовать инструкцию DROP TABLE.

Если таблица содержит столбец идентификаторов, счетчик этого столбца сбрасывается до начального значения, определенного для этого столбца. Если начальное значение не задано, используется значение по умолчанию, равное 1. Чтобы сохранить столбец идентификаторов, используйте инструкцию DELETE.

Ограничения.

Инструкцию TRUNCATE TABLE нельзя использовать для таблиц, для которых выполняются следующие условия:

1) На таблицу ссылается ограничение FOREIGN KEY (таблицу, имеющую внешний ключ, ссылающийся сам на себя, можно усечь.)

2) Таблица является частью индексированного представления.

3) Таблица опубликована с использованием репликации транзакций или репликации слиянием.

Для таблиц с какими-либо из этих характеристик следует использовать инструкцию DELETE.

Инструкция TRUNCATE TABLE не может активировать триггер, поскольку она не записывает в журнал удаление отдельных строк.

4 Функции T-SQL

Функции для работы со строками

LEN: возвращает количество символов в строке. В качестве параметра в функцию передается строка, для которой надо найти длину:
SELECT LEN('Apple') -- 5

LTRIM: удаляет начальные пробелы из строки. В качестве параметра принимает строку:
SELECT LTRIM(' Apple')

RTRIM: удаляет конечные пробелы из строки. В качестве параметра принимает строку:
SELECT RTRIM(' Apple   ')

CHARINDEX: возвращает индекс, по которому находится первое вхождение подстроки в строке. В качестве первого параметра передается подстрока, а в качестве второго - строка, в которой надо вести поиск:
SELECT CHARINDEX('pl', 'Apple') -- 3

PATINDEX: возвращает индекс, по которому находится первое вхождение определенного шаблона в строке:
SELECT PATINDEX('%p_e%', 'Apple')   -- 3

LEFT: вырезает с начала строки определенное количество символов. Первый параметр функции - строка, а второй - количество символов, которые надо вырезать сначала строки:
SELECT LEFT('Apple', 3) – App

RIGHT: вырезает с конца строки определенное количество символов. Первый параметр функции - строка, а второй - количество символов, которые надо вырезать сначала строки:
SELECT RIGHT('Apple', 3)   -- ple

SUBSTRING: вырезает из строки подстроку определенной длиной, начиная с определенного индекса. Певый параметр функции - строка, второй - начальный индекс для вырезки, и третий параметр - количество вырезаемых символов:
SELECT SUBSTRING('Galaxy S8 Plus', 8, 2)   -- S8

REPLACE: заменяет одну подстроку другой в рамках строки. Первый параметр функции - строка, второй - подстрока, которую надо заменить, а третий - подстрока, на которую надо заменить:
SELECT REPLACE('Galaxy S8 Plus', 'S8 Plus', 'Note 8')   -- Galaxy Note 8

REVERSE: переворачивает строку наоборот:
SELECT REVERSE('123456789') – 987654321

CONCAT: объединяет две строки в одну. В качестве параметра принимает от 2-х и более строк, которые надо соединить:
SELECT CONCAT('Tom', ' ', 'Smith') -- Tom Smith

LOWER: переводит строку в нижний регистр:
SELECT LOWER('Apple')   -- apple

UPPER: переводит строку в верхний регистр
SELECT UPPER('Apple') -- APPLE

SPACE: возвращает строку, которая содержит определенное количество пробелов

Строковые функции позволяют производить операции с одной или несколькими строками.
‘Строка1’+ ‘Строка2’ присоединяет Строку1 к Строке2;
ASCII(Char) – возвращает ASCII код с самого левого символа выражения Char;
CHAR(Int) – выводит символ соответствующий ASCII коду в выражении Int;
CHARINDEX(Образец, Выражение) – выводит позицию Образца выражения, то есть где находится Образец в Выражении;
DIFFERENCE(Выражение1, Выражение2) – сравнивает два выражения, выводит числа от 0 до 4: 0 – выражения абсолютно различны; 4 – выражения абсолютно идентичны. Оба выражения типа данных Char;
WCHAR(Int) – выводит выражение Int в формате Unicode;
REPLACE(Строка1, Строка2, Строка3) – меняет в Строке1 все элементы Строка2 на элементы Строка3;
REPLICATE(Char, Int) – повторяет строку Char Int раз;
REVERSE(Char) - производит инверсию строки Char, то есть располагает символы в обратном порядке;
SPACE(Int) – выводит Int пробелов;
STR(Float) – переводит число Float в строку;
STUFF(Выражение1, Начало, Длина, Выражение2) – удаляет из Выражения1 начиная с позиции символа Начало количество символов равное параметру Длина, вместо них подставляет Выражение2;
SUBSTRING(Выражение, Начало, Длина) – из Выражения выводится строка заданной Длины начиная с позиции Начало;
UNICODE(Char) – выводит код в формате Unicode первого символа в строке Char;
LOWERC(Char) – переводит строку Char в маленькие буквы;
UPPER(Char) – переводит строку Char в заглавные буквы.
Примеры применения строковых функций:
SELECT ASCII(‘G’) результат 71.
SELECT LOWER(‘ABC’) результат abc.
SELECT Right(‘ABCDE’) результат CDE
SELECT REVERCE(‘МИР’) результат РИМ.
Замечание. Во всех строковых функциях значения выражения типа Char заключаются в одинарные кавычки.

Функции для работы с числами

ROUND: округляет число. В качестве первого параметра передается число. Второй параметр указывает на длину. Если длина представляет положительное число, то оно указывает, до какой цифры после запятой идет округление. Если длина представляет отрицательное число, то оно указывает, до какой цифры с конца числа до запятой идет округление
SELECT ROUND(1342.345, 2) -- 1342.350
SELECT ROUND(1342.345, -2) -- 1300.000

ISNUMERIC: определяет, является ли значение числом. В качестве параметра функция принимает выражение. Если выражение является числом, то функция возвращает 1. Если не является, то возвращается 0.
SELECT ISNUMERIC(1342.345) -- 1

```
SELECT ISNUMERIC('1342.345')       -- 1
SELECT ISNUMERIC('SQL')        -- 0
SELECT ISNUMERIC('13-04-2017')  -- 0
```

ABS: возвращает абсолютное значение числа.

```
SELECT ABS(-123)    -- 123
```

CEILING: возвращает наименьшее целое число, которое больше или равно текущему значению.

```
SELECT CEILING(-123.45)     -- -123
SELECT CEILING(123.45)      -- 124
```

FLOOR: возвращает наибольшее целое число, которое меньше или равно текущему значению.

```
SELECT FLOOR(-123.45)       -- -124
SELECT FLOOR(123.45)        -- 123
```

SQUARE: возводит число в квадрат.

```
SELECT SQUARE(5)        -- 25
```

SQRT: получает квадратный корень числа.

```
SELECT SQRT(225)        -- 15
```

RAND: генерирует случайное число с плавающей точкой в диапазоне от 0 до 1.

```
SELECT RAND()       -- 0.707365088352935
SELECT RAND()       -- 0.173808327956812
```

COS: возвращает косинус угла, выраженного в радианах

```
SELECT COS(1.0472)  -- 0.5 - 60 градусов
```

SIN: возвращает синус угла, выраженного в радианах

```
SELECT SIN(1.5708)  -- 1 - 90 градусов
```

TAN: возвращает тангенс угла, выраженного в радианах

```
SELECT TAN(0.7854)  -- 1 - 45 градусов
```

DEGREES (Numeric) – преобразует радианы в градусы;
EXP(Float) – экспонента, e^x;
FLOOR(Numeric) – наименьшее целое меньшее или равное выражению numeric;
LOG(Float) – натуральный логарифм ln;
LOG10(Float) – десятичный логарифм $\log_{10}$;
PI() – число пи;
POWER (Numeric,y) – возводит выражение Numeric в степень y;
RADIANS (Numeric) – преобразует градусы в радианы;
RAND () – генерирует случайное число типа данных Float, расположенное между нулем и единицей;
ROUND (Numeric, Длина) – округляет выражение Numeric до заданной Длины (количество знаков после запятой);
SIGN (Numeric) – выводит знак числа +/- или ноль;
SQUARE (Float) – вычисляет квадрат числа Float;
SQRT (Float) – вычисляет квадратный корень числа Float.

Агрегатные функции

Агрегатные функции – позволяют вычислять итоговые значения по полям таблицы.
Выражения 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. Во всех других случаях это недопустимо.

AVGL(поле) – выводит среднее значение поля;
COUNT(*) – подсчитывает общее количество строк, удовлетворяющих условию, включая пустые (NULL);
COUNT(поле) – подсчитывает количество строк с непустым (не NULL) значением указанного столбца;
MAX(поле) – выводит максимальное значение поля;
MIN(поле) – выводит минимальное значение поля;
STDEV(поле) – выводит средне квадратичное отклонение всех значений поля;
STDEVP(поле) – выводит среднеквадратичное отклонение различных значений поля;
SUM(поле) – суммирует все значений поля;
TOP n [Percent] – выводит n первых записей из таблица, либо n% записей из таблицы;
VAR(поле) – выводит дисперсию всех значений поля;
VARP(поле) – выводит дисперсию всех различных значений поля.
Примеры использования агрегатных функций:
SELECT AVG(возраст) FROM Студенты – выводит средний возраст студента из таблицы «Студенты».
SELECT COUNT(ФИО) FROM Студенты – выводит количество различных ФИО из таблицы «Студенты».
SELECT Top 100 FROM Студенты – выводит первые 100 студентов из таблицы «Студенты».__

Функции по работе с датами и временем

GETDATE: возвращает текущую локальную дату и время на основе системных часов в виде объекта datetime
SELECT GETDATE() -- 2017-07-28 21:34:55.830

GETUTCDATE: возвращает текущую локальную дату и время по гринвичу (UTC/GMT) в виде объекта datetime
SELECT GETUTCDATE() -- 2017-07-28 18:34:55.830

SYSDATETIME: возвращает текущую локальную дату и время на основе системных часов, но отличие от GETDATE состоит в том, что дата и время возвращаются в виде объекта datetime2
SELECT SYSDATETIME() -- 2017-07-28 21:02:22.7446744

SYSUTCDATETIME: возвращает текущую локальную дату и время по гринвичу (UTC/GMT) в виде объекта datetime2
SELECT SYSUTCDATETIME() -- 2017-07-28 18:20:27.5202777

SYSDATETIMEOFFSET: возвращает объект datetimeoffset(7), который содержит дату и время относительно GMT
SELECT SYSDATETIMEOFFSET() -- 2017-07-28 21:02:22.7446744 +03:00

DAY: возвращает день даты, который передается в качестве параметра
SELECT DAY(GETDATE()) -- 28

MONTH: возвращает месяц даты
SELECT MONTH(GETDATE()) -- 7

YEAR: возвращает год из даты
SELECT YEAR(GETDATE()) -- 2017

DATENAME: возвращает часть даты в виде строки. Параметр выбора части даты передается в качестве первого параметра, а сама дата передается в качестве второго параметра:
SELECT DATENAME(month, GETDATE()) -- July

Для определения части даты можно использовать следующие параметры (в скобках указаны их сокращенные версии):

year (yy, yyyy): год

quarter (qq, q): квартал

month (mm, m): месяц

dayofyear (dy, y): день года

day (dd, d): день месяца

week (wk, ww): неделя

weekday (dw): день недели

hour (hh): час

minute (mi, n): минута

second (ss, s): секунда

millisecond (ms): миллисекунда

microsecond (mcs): микросекунда

nanosecond (ns): наносекунда

DATEPART: возвращает часть даты в виде числа. Параметр выбора части даты передается в качестве первого параметра (используются те же параметры, что и для DATENAME), а сама дата передается в качестве второго параметра:
SELECT DATEPART(month, GETDATE()) -- 7

DATEADD: возвращает дату, которая является результатом сложения числа к определенному компоненту даты. Первый параметр представляет компонент даты, описанный выше для функции DATENAME. Второй параметр - добавляемое количество. Третий параметр - сама дата, к которой надо сделать прибавление:
SELECT DATEADD(month, 2, '2017-7-28') -- 2017-09-28 00:00:00.000

```
SELECT DATEADD(day, 5, '2017-7-28')     -- 2017-08-02 00:00:00.000
SELECT DATEADD(day, -5, '2017-7-28')        -- 2017-07-23 00:00:00.000
```
Если добавляемое количество представляет отрицательное число, то фактически происходит уменьшение даты.

DATEDIFF: возвращает разницу между двумя датами. Первый параметр - компонент даты, который указывает, в каких единицах стоит измерять разницу. Второй и третий параметры - сравниваемые даты:
```
SELECT DATEDIFF(year, '2017-7-28', '2018-9-28')       -- разница 1 год
SELECT DATEDIFF(month, '2017-7-28', '2018-9-28')    -- разница 14 месяцев
SELECT DATEDIFF(day, '2017-7-28', '2018-9-28')        -- разница 427 дней
```

TODATETIMEOFFSET: возвращает значение datetimeoffset, которое является результатом сложения временного смещения с другим объектом datetimeoffset
```
SELECT TODATETIMEOFFSET('2017-7-28 01:10:22', '+03:00')
```

SWITCHOFFSET: возвращает значение datetimeoffset, которое является результатом сложения временного смещения с объектом datetime2
```
SELECT SWITCHOFFSET(SYSDATETIMEOFFSET(), '+02:30')
```

EOMONTH: возвращает дату последнего дня для месяца, который используется в переданной в качестве параметра дате.
```
SELECT EOMONTH('2017-02-05')    -- 2017-02-28
SELECT EOMONTH('2017-02-05', 3) -- 2017-05-31
```

В качестве необязательного второго параметра можно передавать количество месяцев, которые необходимо прибавить к дате. Тогда последний день месяца будет вычисляться для новой даты.

DATEFROMPARTS: по году, месяцу и дню создает дату
```
SELECT DATEFROMPARTS(2017, 7, 28)       -- 2017-07-28
```

ISDATE: проверяет, является ли выражение датой. Если является, то возвращает 1, иначе возвращает 0.
```
SELECT ISDATE('2017-07-28')     -- 1
SELECT ISDATE('2017-28-07')     -- 0
SELECT ISDATE('28-07-2017')     -- 0
SELECT ISDATE('SQL')            -- 0
```

Системные функции

Системные функции предназначены для получения информации о базе данных и ее содержимом. В SQL сервере существуют следующие системные функции:

COL_LENGTH(таблица, поле) – выводит ширину поля;

DATALENGTH(выражение) – выводит длину выражения;

GETANSINULL(имя БД) – выводит допустимо или недопустимо использовать в БД значение NULL;

IDENTINCR(таблица) – выводит шаг увеличения поля счетчика в таблице;

IDENT_SEED(таблица) – выводит начальное значение счетчиков в таблице;

ISDATE(выражение) – выводит единицу, если выражение является датой и ноль, если не является;

ISNUMERIC(выражение) – выводит единицу, если выражение является числовым и ноль, если не числовым;

NULIFF(выражение1, выражение2) – выводит ноль если выражение1 равно выражению 2.

STR – используется для конвертирования значений числовых типов данных в символьные строки

STR (*float_expression* [, *length* [, *decimal*]])

float_expression – выражение типа данных float с десятичной запятой.

length - общая длина. Она включает десятичную запятую, знак, цифры и пробелы. Значение по умолчанию 10.

decimal - количество знаков справа от десятичной запятой. Аргумент *decimal* должен быть меньше или равен 16.

Для выполнения других преобразований в SQL сервере используются функции преобразования исходной величины в конечный (целевой) тип данных. такие, как CAST, CONVERT, PARSE или TRY_CAST, TRY_CONVERT и TRY_PARSE.
Функция CAST является стандартной, функция CONVERT – нет. Нестандартную функцию CONVERT следует выбирать только тогда, когда необходимо использование аргумента стиля.

Синтаксис стандартной функции CAST:

CAST (expression AS data_type [(length)])

Синтаксис нестандартной функции CONVERT:

CONVERT (data_type [(length)] , expression [, style])

Аргументом expression функций является величина, которую необходимо конвертировать.

length – длина целевого типа

аргумент style позволяет можно управлять стилем представления значений datetime, smalldatetime, float, real, money и smallmoney при конвертировании их в символьные строки (nchar, nvarchar, char и varchar).

Разница между функциями без части TRY и их аналогами с этой частью состоит в том, что функции без TRY завершаются ошибкой, если значение невозможно преобразовать, тогда как функции с TRY возвращают в таком случае NULL. Например, следующий код завершится ошибкой:

SELECT CAST('abc' AS INT);

В свою очередь, этот код возвратит значение NULL:

SELECT TRY_CAST('abc' AS INT);

COALESCE (a1, a2) Возвращает первое значение выражения из списка выражений a1, a2, ..., которое не является значением null.

COL_LENGTH (obj, col) Возвращает длину столбца col объекта базы данных (таблицы или представления) obj.

CURRENT_TIMESTAMP Возвращает текущие дату и время.

CURRENT_USER Возвращает имя текущего пользователя.

DATALENGTH (z) Возвращает число байтов, которые занимает выражение z.

GETANSINULL ('dbname') Возвращает 1, если использование значений null в базе данных dbname отвечает требованиям стандарта ANSI SQL.

ISNULL (expr, value) Возвращает значение выражения expr, если оно не равно NULL; в противном случае возвращается значение value.

ISNUMERIC (expr) Определяет, имеет ли выражение expr действительный числовой тип.

NEWID() Создает однозначный идентификационный номер ID, состоящий из 16-байтовой двоичной строки, предназначенной для хранения значений типа данных UNIQUEIDENTIFIER.

NEWSEQUENTIALID() Создает идентификатор GUID, больший, чем любой другой идентификатор GUID, созданный ранее этой функцией на указанном компьютере. (Эту функцию можно использовать только как значение по умолчанию для столбца.)

NULLIF (expr1, expr2) Возвращает значение null, если значения выражений expr1 и expr2 одинаковые.

SERVERPROPERTY (propertyname) Возвращает информацию о свойствах сервера базы данных.

SYSTEM_USER Возвращает ID текущего пользователя.
USER_ID ([username]) Возвращает идентификатор пользователя username. Если пользователь не указан, то возвращается идентификатор текущего пользователя.

USER_NAME ([id]) Возвращает имя пользователя с указанным идентификатором id. Если идентификатор не указан, то возвращается имя текущего пользователя.

Функции метаданных Transact-SQL

Функции метаданных возвращают информацию об указанной базе данных и объектах базы данных.

COL_NAME (tab_id, col_id) Возвращает имя столбца с указанным идентификатором col_id таблицы с идентификатором tab_id.

COLUMNPROPERTY (id, col, property) Возвращает информацию об указанном столбце.

DATABASEPROPERTY (database, property) Возвращает значение свойства property базы данных database.

DB_ID ([db_name]) Возвращает идентификатор базы данных db_name. Если имя базы данных не указано, то возвращается идентификатор текущей базы данных.

DB_NAME ([db_id]) Возвращает имя базы данных, имеющей идентификатор db_id. Если идентификатор не указан, то возвращается имя текущей базы данных.

INDEX_COL (table, i, no) Возвращает имя индексированного столбца таблицы table. Столбец указывается идентификатором индекса i и позицией no столбца в этом индексе.

INDEXPROPERTY (obj_id, index_name, property) Возвращает свойства именованного индекса или статистики для указанного идентификационного номера таблицы, имя индекса или статистики, а также имя свойства.

OBJECT_NAME (obj_id) Возвращает имя объекта базы данных, имеющего идентификатор obj_id.

OBJECT_ID (obj_name) Возвращает идентификатор объекта obj_name базы данных.

OBJECTPROPERTY (obj_id, property) Возвращает информацию об объектах из текущей базы данных.

Задание

Необходимо для разрабатываемой БД написать по три команды INSERT, UPDATE, DELETE для каждой таблицы (с использованием различных функций).

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

Контрольные вопросы

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

2 Привести примеры использования команд INSERT, UPDATE, DELETE.

3 Указать назначение, преимущества и ограничения инструкции TRUNCATE TABLE.

Лабораторная работа № 19. Создание индексов средствами SQL

2 часа

Цель: получение навыков определения индексируемых колонок таблиц.

19.1 Теоретические положения

Индекс представляет собой дополнение к таблице, помогающее ускорить поиск необходимых данных за счет физического или логического их упорядочивания. Индекс является набором ссылок, упорядоченным по определенной (индексируемой) колонке таблицы. Физически индекс представляет собой упорядоченный набор значений из индексированной колонки 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.

Различают кластерный, некластерный и уникальный индексы. Более подробно индексы описаны в конспекте лекций и в [4, 13, 15].

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

Ключевые колонки — это такие колонки, которые задают критерии выборки данных, например порядок сортировки. Не стоит индексировать колонки, 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 такой колонки, выбрав из нее до десяти первых символов, и индексировать ее. Индексирование длинных колонок может существенно снизить производительность работы сервера. Индекс является самостоятельным объектом базы данных, но он связан с определенной колонкой таблицы. Работа индексов базируется на возможности уникально идентифицировать строку в таблице. Именно эта возможность обеспечивает быстрый поиск нужных данных.

Кластерный индекс

Принципиальным отличием кластерного индекса (Clustered Index) от индексов других типов является то, что при его определении в таблице физическое расположение данных перестраивается в соответствии со структурой индекса. Информация об индексе и сами данные физически располагаются вместе

Использование кластерных индексов способно существенно увеличить производительность поиска данных даже по сравнению с обычными индексами. Если в таблице определен не кластерный индекс, то сервер должен сначала обратиться к индексу, а затем найти нужную строку в таблице. При использовании кластерных индексов следующая порция данных располагается сразу же

после найденных ранее данных. Благодаря этому отпадают лишние операции обращения к индексу и нового поиска нужной строки в таблице.

Естественно, в таблице может быть определен только один кластерный индекс. В качестве кластерного индекса следует выбирать наиболее часто используемые колонки. Следует избегать создания кластерного индекса для часто изменяемых колонок, так как сервер должен будет выполнять физическое перемещение всех данных в таблице, чтобы они находились в упорядоченном состоянии, как того требует кластерный индекс.

Кластерный индекс может включать несколько колонок. Но количество колонок кластерного индекса следует по возможности свести к минимуму, так как все некластерные индексы, созданные в этой же таблице, используют кластерный индекс для уникальной идентификации строки в таблице.

Некластерный индекс

Некластерные индексы являются наиболее типичными представителями семейства индексов. В отличие от кластерных, они не перестраивают физическую структуру таблицы, а лишь организуют ссылки на соответствующие строки.

Для идентификации нужной строки в таблице некластерный индекс организует специальные указатели (row locator). Эти указатели содержат информацию об идентификационном номере фата (ID file), в котором хранится строка, а также об идентификационном номере страницы и номере искомой строки на этой странице. Если же в таблице определен кластерный индекс, то указатель ссылается не на физическое положение строки в базе данных, а на соответствующий элемент кластерного индекса, описывающего эту строку. Это позволяет не перестраивать структуру некластерных индексов всякий раз, когда кластерный индекс меняет физический порядок строк в таблице. Изменяется только кластерный индекс, а некластерные индексы обновляют только индексируемое значение, но не указатель.

Если при построении некластерного индекса кластерный индекс был не уникален, то SQL Server автоматически добавляет к нему дополнительные значения, которые делают его уникальным. Для пользователя эти дополнительные значения не видны, и он может работать с кластерным индексом как обычно.

В одной таблице можно определить до 249 некластерных индексов. Однако в большинстве случаев следует ограничиться 4-5 индексами.

Уникальный индекс

Уникальные индексы (Unique Indexes) гарантируют уникальность значений в индексируемой колонке. Сервер не разрешит вставить новое или изменить существующее значение таким образом, что в результате этой операции в колонке будет существовать два одинаковых значения.

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.

Кластерные индексы используются только тогда, когда это действительно необходимо. Для обеспечения целостности данных в колонке можно определить ограничение целостности UNIQUE или PRIMARY KEY, а не прибегать к использованию уникальных индексов. Использование уникальных индексов только для обеспечения целостности данных является неоправданной тратой пространства в базе данных.

Формат команды CREATE INDEX на T-SQL имеет вид:

```
CREATE [UNIQUE] [CLUSTERED | NONCLUSTERED]
INDEX index_name ON table (column [...n])
```

UNIQUE – при указании этого ключевого слова будет создан уникальный индекс. При создании такого индекса сервер выполняет предварительную проверку колонки на уникальность значений. Если в колонке есть хотя бы два одинаковых значения, индекс не создается и сервер выдает сообщение об ошибке. В индексируемой колонке также желательно запретить хранение значений NULL, чтобы избежать проблем, связанных с уникальностью значений. После того как для колонки создан уникальный индекс, сервер не разрешает выполнение команд INSERT и UPDATE, которые приведут к появлению дублирующихся значений.

CLUSTERED – создаваемый индекс будет кластерным, то есть физически данные будут располагаться в порядке, определяемом этим индексом. Кластерным может быть только один индекс в таблице.

NONCLUSTERED – создаваемый индекс будет некластерным. В таблице можно определить до 249 некластерных индексов.

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

table (column [...n]) – имя таблицы, в которой содержатся одна или несколько индексируемых колонок. В скобках указываются имена колонок, на основе которых будет построен индекс. Не допускается построение индекса на основе колонок с типом данных text, ntext, image или bit. Если указывается несколько колонок, то создаваемый индекс будет смешанным (composite index). В один смешанный индекс можно включить до 16 колонок.

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 колонки 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?

Лабораторная работа № 20. Взаимодействие СУБД MS SQL Server с системой программирования MS Visual Studio.NET

2 часа

Цель: изучение основ создания приложения Windows Presentation Foundation (WPF) или Windows Forms в среде Visual Studio .Net для ввода, изменения и удаления данных в базе данных.

20.1 Теоретические положения

Вопросы, изучаемые в данной теме, описаны в [5, 16, 17]. Рекомендуется также использовать следующие материалы:

https://metanit.com/sharp/tutorial/

https://metanit.com/sharp/wpf/1.php

https://docs.microsoft.com/ru-ru/dotnet/framework/wpf/getting-started/walkthrough-my-first-wpf-desktop-application

Задание

Необходимо создать приложение Windows Presentation Foundation (WPF) или Windows Forms в среде Visual Studio .Net или UWP «Универсальная платформа Windows» (Universal Windows Platform, UWP) для ввода, изменения и удаления данных в таблицах базы данных.

Содержание отчета: тема и цель работы; прокомментированный код на C# и скриншоты приложения.

Контрольные вопросы

1 Как установить соединение с БД?

2 Как выполняется удаление и обновление записей в Windows-формах?

Лабораторная работа № 21. Назначение прав доступа пользователям к объектам базы данных средствами T-SQL

2 часа

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

21.1 Теоретические положения

Вопросы управления правами доступа к объектам БД подробно описаны в конспекте лекций и в [15]. Здесь же рассмотрен SQL-код основных инструкций.

Для предоставления прав доступа используется команда GRANT:

```
GRANT
{ ALL / permissions [..n]}
{ [ (column [..n] )] ON
{ table / view }
| ON { table / view} [ (column[..n]) ]
| ON {stored_procedure}
TO security_account
[WITH GRANT OPTION]  [AS {group / role}]
```

Для запрещения пользователям доступа к объектам БД используется команда DENY:

```
DENY
{ ALL / permissions [..n]}
{ [ (column [..n] )] ON
{ table / view }
| ON { table/view } [(column[..n])]
| ON {stored_procedure}
TO security_account [..n]  [CASCADE]
```

Для неявного отклонения доступа к объектам БД используется команда REVOKE.

```
REVOKE
{ ALL / permissions [..n] }
{ [ (column [..n])] ON
{ table / view }
| ON { table / view } [(column[..n])]
| ON { stored_procedure }
TO security_account [..n]  [ CASCADE ]  [ AS { role / group} ]
```

Значения параметров команд:

ALL – означает, что пользователю будут предоставлены все возможные разрешения;

Permissions – список доступных операций, которые предоставляются пользователю. Можно предоставлять одновременно несколько разрешений;

Statement – предоставление разрешений на выполнение команд T-SQL;

Security_account – указывается имя объекта системы безопасности, которую необходимо включить в роль. Это могут быть как учетные записи SQL-сервер, так и группы пользователей Windows;

With Grant Option – позволяет пользователю, которому предоставляются права, назначать права на доступ к объекту для других пользователей;

As role / group – позволяет указать участие пользователя в роли, которому предоставляется возможность предоставлять права другим пользователям;

CASCADE – позволяет отзывать права не только у данного пользователя, но и у всех пользователей, которым он предоставил данные права.

Задание

Для разрабатываемой БД необходимо реализовать систему безопасности: создать 5 пользователей и назначить им права доступа к таблицам, представлениям и хранимым процедурам.

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

Контрольные вопросы

1 На какие категории можно разделить права в SQL-сервер?

2 Перечислить стандартные роли сервера и базы данных.

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

Лабораторная работа № 22. Работа с базами данных с использованием технологии ADO.NET

2 часа

Цель: изучить основы технологии ADO.NET и компоненты доступа к данным в MS Visual Studio .Net.

22.1 Теоретические положения

ADO.NET – это набор средств Microsoft .NET Framework, позволяющих приложению легко управлять и взаимодействовать со своим файловым или серверным хранилищем данных.

Для работы с различными СУБД подключаются (using) соответствующие пространства имен: для MS Access – System.Data.OleDb; для MS SQL – System.Data.SqlClient.

В объектной модели ADO.NET можно выделить несколько уровней.

Уровень данных. Это по сути дела базовый уровень, на котором располагаются сами данные (например, таблицы БД MS SQL Server). На данном уровне обеспечивается физическое хранение информации на магнитных носителях и манипуляция с данными на уровне исходных таблиц (выборка, сортировка, добавление, удаление, обновление и т. п.).

Уровень бизнес-логики. Это набор объектов, определяющих, с какой БД предстоит установить связь и какие действия необходимо будет выполнить с содержащейся в ней информацией. Для установления связи с БД используется объект DataConnection. Для хранения команд, выполняющих какие либо действия над данными, используется объект DataAdapter. И, наконец, если выполнялся процесс выборки информации из БД, для хранения результатов выборки используется объект DataSet. Объект DataSet представляет собой набор данных «вырезанных» из таблиц основного хранилища, который может быть передан любой программе-клиенту, способной отобразить эту информацию пользователю, либо выполнить какие-либо манипуляции с полученными данными.

Каждый объект DataAdapter обеспечивает обмен данными между одной таблицей источника данных (базы данных) и одним объектом DataTable в наборе данных DataSet. Если DataSet сдержит несколько таблиц (объектов DataTable), то необходимо иметь и несколько адаптеров данных.

Используя объект DataAdapter, можно читать, добавлять, модифицировать и удалять записи в источнике данных. Чтобы определить, как каждая из этих операций должна произойти, DataAdapter поддерживает следующие свойства: SelectCommand – описание команды, которая обеспечивает выборку нужной информации из базы данных; InsertCommand – описание команды, которая обеспечивает добавление записей в базу данных; UpdateCommand – описание команды, которая обеспечивает обновление записей в базе данных;

DeleteCommand – описание команды, которая обеспечивает удаление записей из базы данных. Каждая из этих команд реализована в виде SQL-запроса или хранимой процедуры. Эти свойства являются самостоятельными объектами и относятся к элементам класса OleDbCommand или SqlCommand. Данные объекты поддерживают свойство CommandText, содержащее описание SQL-запроса или хранимой процедуры.

Уровень приложения. Это набор объектов, позволяющих хранить и отображать данные на компьютере конечного пользователя. Для хранения информации используется объект DataSet, а для отображения данных имеется набор элементов управления (DataGrid, TextBox, ComboBox, Label и т. д.).

Обмен данными между приложениями и уровнем бизнес-логики происходит с использованием формата XML, а средой передачи данных служат либо локальная сеть (Интранет), либо глобальная сеть (Интернет).

В ADO.NET для манипуляции с данными могут использоваться команды, реализованные в виде SQL-запросов или хранимых процедур (DataCommand). Например, если нужно получить некий набор информации БД, формируется команда SELECT или вызывается хранимая процедура по ее имени.

Когда требуется получить набор строк из БД, необходимо выполнить следующую последовательность действий:

– открыть соединение (connection) с БД;

– вызвать на исполнение метод или команду, указав ей в качестве параметра текст SQL-запроса или имя хранимой процедуры;

– закрыть соединение с базой данных.

Связь с базой данных остается активной только на достаточно короткий срок – на период выполнения запроса или хранимой процедуры.

Когда команда вызывается на исполнение, она возвращает либо данные, либо код ошибки. Если в команде содержался SQL-запрос на выборку SELECT, то команда может вернуть набор данных. Можно выбрать из БД только определенные строки и колонки, используя объект DataReader, который работает достаточно быстро, т. к. использует курсоры read-only, forward-only.

Для считывания данных, хранящихся в таблице, используется метод ExecuteReader(). Этот метод возвращает объект SqlDataReader, который используется для чтения данных.

Для определения параметров запросов используется объект SqlParameter. Параметры, которые используются в командах, могут быть нескольких типов. Тип параметра задается с помощью свойства Direction объкта SqlParameter. Данное свойство принимает одно из значений перечисления ParameterDirection: Input, InputOutput, Output, ReturnValue.

По сравнению с запросами, которые посылаются из приложения базе данных, хранимые процедуры определяются на сервере и предоставляют большую производительность и являются более безопасными.

Объект SqlCommand имеет встроенную поддержку хранимых процедур. В частности у него определено свойство CommandType, которое в качестве значения принимает значение из перечисления System.Data.CommandType. И зна-

чение System.Data.CommandType.StoredProcedure как раз указывает, что будет использоваться хранимая процедура.

При выполнении лабораторной работы рекомендуется также использовать https://metanit.com/sharp/adonet/1.1.php

Задание

Необходимо, используя технологию ADO.NET, реализовать вызов хранимых процедур и вывод результатов работы запросов в пользовательском интерфейсе.

Содержание отчета: тема и цель работы; код C# подключения.

Контрольные вопросы

1 Как осуществляется взаимодействие с БД через объект DataSet?

2 Для чего предназначены объекты SqlConnection, OleDbConnection, DataAdapter DataReader?

Лабораторная работа № 23. Работа с базами данных с использованием технологии ASP.NET

4 часа

Цель: изучение основ использования технологии ASP.NET при работе с БД.

23.1 Теоретические положения

Вопросы, изучаемые в данной теме, описаны в [17, 18, 18]. Рекомендуется также использовать следующие материалы:
https://metanit.com/sharp/mvc.php

Задание

Необходимо создать web-форму для доступа к таблицам базы данных с использованием технологии ASP.NET MVC или ASP.NET Core.

Содержание отчета: тема и цель работы; код C# выполнения задания.

Контрольные вопросы

1 Охарактеризовать суть паттерна MVC.
2 Какова структура *ASP.NET* приложения?

Список литературы

1 ГОСТ 34.602-89 «Информационная технология. Комплекс стандартов на автоматизированные системы. Техническое задание на создание автоматизированной системы» [Электронный ресурс]. – Москва : Стандартинформ, 2009 (© АО «Кодекс», 2018). – Режим доступа : http://docs.cntd.ru/document/gost-34-602-89. – Дата доступа : 29.03.2018.

2 **Агальцов В. П.** Базы данных : учебник для вузов: в 2 кн. Кн. 1 : Локальные базы данных / В. П. Агальцов. – 2-е изд., перераб. и доп. – Москва : Форум : Инфра-М, 2012. – 352 с.

3 **Агальцов, В. П.** Базы данных. В 2-х т. Т. 2. Распределенные и удаленные базы данных : учебник / В.П. Агальцов. – Москва : Форум : Инфра-М, 2015. – 272 с. : ил.

4 **Гросс, К.** C# 2008 и платформа NET 3.5 Framework : вводный курс : [Пер. с англ.] / К. Гросс. – 2-е изд. – Москва : И. Д. Вильямс, 2009. – 480 с.

5 **Кузин, А. В.** Базы данных : учеб. пособие для студентов высших учебных заведений / А. В. Кузин, С. В. Левонисова. – 6-е изд., стер. – Москва : Издательский центр «Академия», 2016. – 320 с.

6 **Кузнецов, С. Д.** Базы данных : учебник / С. Д. Кузнецов. – Москва : Академия, 2012. – 496 с.

7 **Эспозито, Д.** Программирование на основе Microsoft ASP.NET MVC : пер. с англ. / Д. Эспозито. – 2-е изд. – Москва : Русская редакция ; Санкт-Петербург : БХВ-Петербург, 2012. – 464 с. : ил.

8 **Олейник, П. П.** Корпоративные информационные системы : учебник / П. П. Олейник. – Санкт-Петербург : Питер, 2012. – 176 с.

9 **Орин, Т.** Оптимизация и администрирование баз данных Microsoft SQL Server 2005. Учебный курс Microsoft : Пер. с англ. / Т. Орин, М. Йен. – Москва : Русская Редакция, 2007. – 624 с.

10 РД IDEF0-2000. Методология функционального моделирования IDEF0. Руководящий документ. – Москва : Изд-во стандартов, 2000. – 75 с.

11 **Советов, Б. Я.** Базы данных: теория и практика : учебник для бакалавров. – 2-е изд. – Москва : Юрайт, 2012. – 463 с.

12 **Черемных, С. В.** Моделирование и анализ систем: IDEF-технологии : Практикум / С. В. Черемных, В. С. Ручкин, И. О. Семенов. – Москва : Финансы и статистика, 2005. – 192 с.

13 **Шустова, Л. И.** Базы данных : учебник / Л. И. Шустова, О. В. Тараканов. – Москва : Инфра-М, 2017. – 304 с. + Доп. материалы [Электронный ресурс. – Режим доступа : http://www.znanium.com]. – (Высшее образование: Бакалавриат). www.dx.doi.org/10.12737/11549.– Дата доступа : 29.03.2018.

14 **Рихтер, Д.** CLR via C#. Программирование на платформе Microsoft .NET Framework 4.5 на языке C# / Д. Рихтер ; пер. с англ. Е. Матвеев. – 4-е изд. – Санкт-Петербург : Питер, 2016. – 896 с. : ил.

15 **Прайс, М. Дж**. C# 7 и .NET Core. Кросс-платформенная разработка для профессионалов. 3-е изд. – Санкт-Петербург : Питер, 2018. – 640 с. : ил.

16 **Фримен, А**. ASP.NET Core MVC с примерами на C# для профессионалов / А. Фримен. – 6-е изд. : Пер. с англ. – Санкт-Петербург : ООО "Альфа-книга", 2017. – 992 с. : ил.

17 ASP.NET Core. Разработка приложений / Дж. Чамберс, Д. Пэкетт, С. Тиммс. – Санкт-Петербург : Питер, 2018. – 464 с.: ил.

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

19 **Шумаков, П. В.** Программирование на языке Transact SQL при работе с СУБД Micriosoft SQL Server / П. В. Шумаков. – М. : АСУ-61, 2019. – 772 с.

20 **Куликов, С. С.** Реляционные базы данных в примерах : практическое пособие для программистов и тестировщиков / С. С. Куликов. – Минск: Четыре четверти, 2020. – 424 с. [Электронный ресурс]. – Режим доступа : http://svyatoslav.biz/relational_databases_book/. – Дата доступа : 29.12.2020.

21 **Куликов, С. С.** Работа с MySQL, MS SQL Server и Oracle в примерах : практ. пособие. / С. С. Куликов. – Минск : БОФФ, 2016. – 556 с. [Электронный ресурс]. – Режим доступа : http://svyatoslav.biz/database_book/. – Дата доступа : 29.12.2020.