Министерство образования Республики Беларусь
Учреждение образования «Витебский государственный
университет имени П.М. Машерова»
Кафедра информационных технологий и управления бизнесом

Н.Д. Адаменко

БАЗЫ ДАННЫХ

***Методические рекомендации
к выполнению лабораторных работ***

Витебск
ВГУ имени П.М. Машерова
2023

УДК 004.65(076.5)
ББК 16.35я73
А28

Печатается по решению научно-методического совета учреждения образования «Витебский государственный университет имени П.М. Машерова». Протокол № 7 от 26.04.2023.

Автор: доцент кафедры информационных технологий и управления бизнесом ВГУ имени П.М. Машерова, кандидат педагогических наук, доцент **Н.Д. Адаменко**

Р е ц е н з е н т ы :
заведующий кафедрой информационных систем
и технологий УО «ВГТУ», кандидат технических наук,
доцент *В.Е. Казаков;*
доцент кафедры прикладного и системного программирования
ВГУ имени П.М. Машерова, кандидат физико-математических наук,
доцент *С.А. Ермоченко*

Адаменко, Н.Д.

А28 Базы данных : методические рекомендации к выполнению лабораторных работ / Н.Д. Адаменко. – Витебск : ВГУ имени П.М. Машерова, 2023. – 55 с.

Предлагаемое учебное издание содержит методические материалы для проведения лабораторных занятий, последовательно формирующих основные умения, необходимые для разработки и создания баз данных. Оно может найти применение при изучении дисциплины «Базы данных» для специальности «Информационные системы и технологии (в здравоохранении)».

УДК 004.65(076.5)
ББК 16.35я73

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

В настоящее время автоматизированные информационные системы создаются практически для всех предприятий, фирм и организаций различных форм собственности, а база данных является ядром любой информационной системы.

Качество разработки и проектирования базы данных в значительной степени определяет эффективность функционирования информационной системы, создаваемой на ее основе. В этой связи актуальной задачей является приобретение студентами опыта проектирования и создания баз данных. В предлагаемом учебном издании рассмотрены теоретические основы и методы, на которых базируется практика разработки реляционных баз данных. Основное внимание уделено способам создания объектов базы – таблиц, запросов, хранимых процедур и триггеров.

Методические рекомендации содержат систему лабораторных работ, последовательное выполнение которых обеспечивает формирование устойчивых умений проектирования и создания баз данных. В технологии разработки многопользовательских информационных систем значительное внимание уделяется безопасности данных при их совместной обработке, поэтому автором освещаются вопросы администрирования баз данных.

Каждая лабораторная работа включает теоретическую часть и систему заданий, выполнение которых способствует формированию базовых компетенций, необходимых разработчику баз данных. Издание может быть использовано для самостоятельного освоения способов проектирования и создания баз данных. В этом случае необходимо усвоить теоретический материал, а затем выполнить один или несколько вариантов заданий.

ЛАБОРАТОРНАЯ РАБОТА № 1
СОЗДАНИЕ БАЗЫ ДАННЫХ. ТИПЫ ДАННЫХ, ИСПОЛЬЗУЕМЫЕ В SQL-СЕРВЕРЕ. СОЗДАНИЕ ПОЛЬЗОВАТЕЛЬСКИХ ТИПОВ ДАННЫХ

Цель работы: изучить способы создания баз данных, рассмотреть основные типы данных и способы создания пользовательских типов данных.

ОСНОВНЫЕ СВЕДЕНИЯ

Создание базы данных

База данных (БД) – некоторый набор таблиц, связанных между собой определённой связью для выполнения различного рода задач.

Создание базы данных в системе SQL-сервер может осуществляться следующими способами: используя стандартные команды языка T-SQL и MS SQL Server Management Studio.

Первый способ: с помощью оператора CREATE DATABASE. В процессе создания базы также создаётся журнал транзакций (с помощью оператора LOG ON).

Создание базы данных – это процесс указания имени базы и определения размеров и размещения файлов базы данных (первичного и вторичных файлов базы данных, файла журнала транзакций). В primary файле базы данных (расширение .mdf) записывается информация об основных её объектах – таблицах, индексах и т.д., а в файл журнала транзакций (расширение .ldf) информация о процессе работы с транзакциями (контроль целостности данных, состояние базы данных до и после выполнения транзакции). Если в процессе использования базы данных планируется размещение её на нескольких дисках, то в этом случае создаются secondary файлы (расширение .ndf). По умолчанию базы данных имеют право создавать только те пользователи, которым назначены роли *sysadmin* и *dbcreator*.

Синтаксис:

CREATE DATABASE *имя_базы_данных*
[ON
[PRIMARY] (NAME=*логическое_имя_файла,*
FILENAME=*'физическое_имя_файла'*
[, SIZE=*размер*]
[, MAXSIZE=*максимальный размер*]
[, FILEGROWTH=*шаг_приращения_размера*])
[, *...n*]
]
[LOG ON
(NAME=*логическое_имя_файла_журнала,*

FILENAME=*'физическое_имя_файла_журнала'*
[, SIZE=*размер_журнала*])
[, *...n*]
]
[FOR RESTORE]

При этом следует помнить, что пробелы используются для разделения элементов команды SQL, и поэтому они не могут быть частью имени базы данных, таблицы или какого-либо другого создаваемого объекта.

При создании базы данных можно указать следующие параметры:

- PRIMARY указывает файлы основной группы файлов, которая содержит все системные таблицы базы данных. Кроме того, здесь также содержатся объекты, не привязанные к пользовательским группам файлов. В любой базе данных должен быть лишь один основной (первичный) файл данных. Он служит отправной точкой базы данных и указывает на все прочие её файлы. Стандартное расширение имени основного файла данных – .mdf. Если ключевое слово PRIMARY опущено, основным файлом становится первый файл в операторе;
- FILENAME – задаёт физическое имя и путь к файлу. В пути (*физическое_имя_файла*) необходимо указывать папку локального диска сервера, на котором установлен SQL Server;
- SIZE – указывает размер файла: в мегабайтах, тогда используется суффикс MB (по умолчанию), или в килобайтах – в этом случае применяется суффикс KB. Минимально возможное значение – 512 кб. Параметр *size* задает минимальный (начальный) размер файла. Файл может увеличиваться, однако его нельзя сжать так, чтобы его объем стал меньше заданного минимального размера;
- MAXSIZE – указывает максимальный размер файла. Если размер не указан, то файл будет увеличиваться до полного заполнения диска;
- FILEGROWTH – задаёт шаг приращения размера файла. Если SQL Server необходимо увеличить размер файла, он увеличит его на значение, заданное параметром FILEGROWTH, причём ноль означает запрет увеличения размера. По умолчанию (если параметр FILEGROWTH не определён) – шаг приращения равен 10%, а его минимальное значение – 64 Кб. Указанный вами размер округляется до ближайшего числа, кратного 64 Кб;
- FOR RESTORE – задаёт восстановление системы по журналу транзакций в случае её сбоя. Имеется в виду сбой системы, нарушающий все выполняемые в данный момент транзакции, но не нарушающий базу данных физически. При сбое носителей, который представляет собой физическую угрозу для данных, восстановление осуществляется с резервной копии БД.

```
CREATE DATABASE [DreamHome]
 ON  PRIMARY
( NAME = N'[DreamHome]',
 FILENAME = N'C:\Program Files\Microsoft SQL
Server\MSSQL15.SQLEXPRESS\MSSQL\DATA\[DreamHome].mdf',
 SIZE = 15MB ,
 MAXSIZE = 30Mb,
FILEGROWTH = 1000KB )
 LOG ON
( NAME = N'[DreamHome]_log',
FILENAME = N'C:\Program Files\Microsoft SQL
Server\MSSQL15.SQLEXPRESS\MSSQL\DATA\[DreamHome]_log.ldf' ,
 SIZE = 3MB ,
 MAXSIZE = 15Mb ,
 FILEGROWTH = 10KB )
GO
```

Рисунок 1

Кроме перечисленных, при создании базы данных можно указать параметры настройки и параметры доступа к базе данных. Например, можно сделать базу данных доступной только для чтения или указать, чтобы регистрационные записи удалялись из журнала транзакций после записи изменённых страниц данных на диск. Можно также осуществить настройку доступа пользователей к базе данных. По умолчанию свойства доступа могут устанавливаться для пользователей, осуществляющих подключение, используя роль *public*. Наиболее часто используемые параметры смотрите в Приложении 2.

Упражнение: При помощи оператора CREATE DATABASE создайте базу данных DreamHome. Параметры базы данных указаны в таблице:

Файл	**Логическое имя**	**Физическое имя**	**Начальный размер**	**Шаг приращения размера**	**Максимальный размер**
База данных	**DreamHome**	C:\Program files\Microsoft Sql server\mssql.1\mssql\ Data\DreamHome.mdf	3 Мб	1 Мб	15 Мб
Журнал	DreamHome_ log	c:\program files\microsoft sql server\mssql.1\mssql\data \DreamHome.ldf	1 Мб	1 Мб	10 Мб

1. Для начала необходимо запустить среду разработки "MS SQL Server Management Studio". Для этого в меню **"Пуск"** выбрать пункт **"Программы\ Microsoft SQL Server 2019\Microsoft SQL Server Tools 18\ Microsoft SQL Server Management Studio"**.
2. Создайте новый запрос, нажав на панели инструментов кнопку ”Создать запрос” .
3. Выполните оператор CREATE DATABASE, чтобы создать базу данных (см. рисунок 1).

4. Просмотрите свойства базы данных с помощью группы процедур *sp_helpdb*, чтобы убедиться, что база создана должным образом:

EXEC sp_helpdb DreamHome

Второй способ:

В обозревателе объектов откройте контекстное **меню** папки Базы данных и в появившемся меню выберите пункт **"Создать базу данных"**.

Появится окно настроек параметров файла данных новой БД (см. рисунок 2)

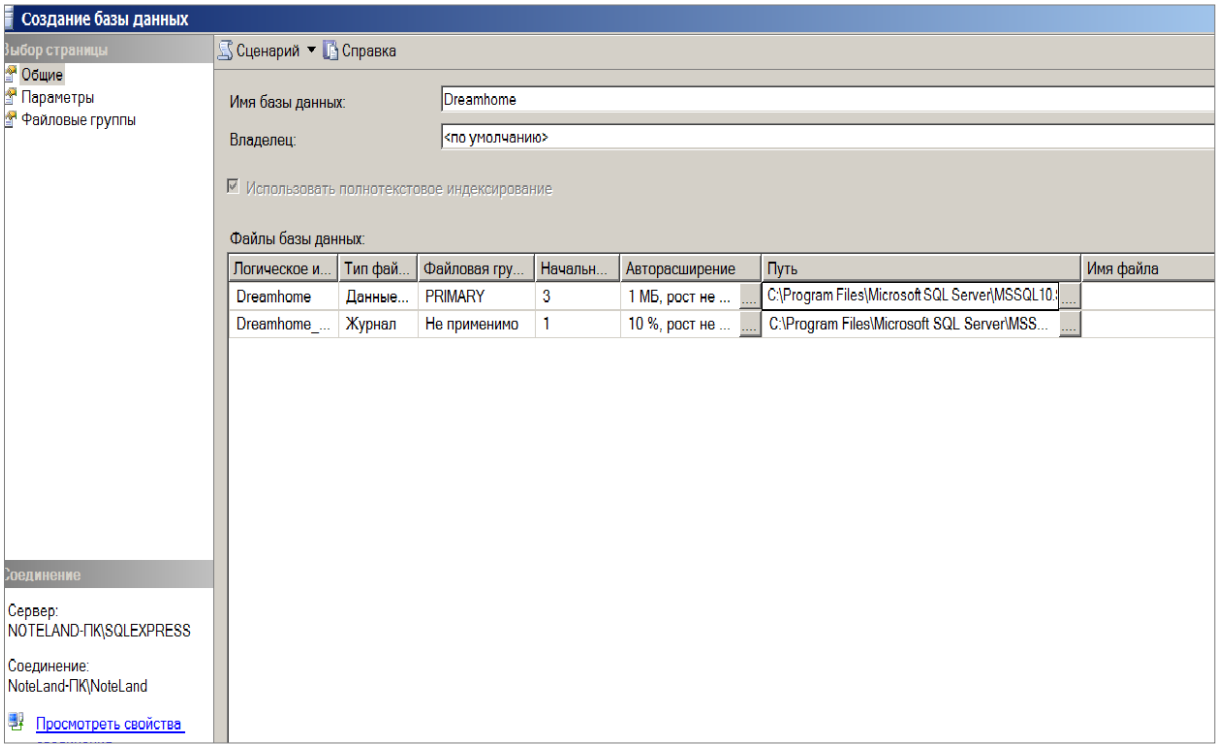


Рисунок 2

На панели **Выбор страницы** выбрать **Общие**.

В верхней части окна расположено два параметра: **Имя Базы данных** и **Владелец**). Задайте **Имя Базы данных - DreamHome**. Параметр **Владелец "** оставить без изменений.

Под вышеприведёнными параметрами в виде таблицы располагаются настройки файла данных и журнала транзакций. Таблица имеет следующие столбцы:

- Логическое имя файла данных и журнала транзакций. По этим именам будет происходить обращение к вышеприведённым файлам в БД. Можно заметить, что файл данных имеет то же имя что и БД, а имя файла журнала транзакций составлено из имени БД и суффикса "_log".
- Тип файла. Этот параметр показывает, является ли файл файлом данных или журналом транзакций.

- Файловая группа показывает, к какой группе файлов относится файл. **Группы файлов** настраиваются в группе настроек **Файловые группы**.
- Начальный размер файла данных и журнала транзакций в мегабайтах.
- Авторасширение размера файла. Как только файл заполняется информацией, его размер автоматически увеличивается на величину, указанную в параметре **Авторасширение**. Увеличение можно задавать как в мегабайтах так и в процентах. Здесь же можно задать максимальный размер файлов. Для изменения этого параметра надо нажать кнопку **"…"**.
- Путь к папке, где хранятся файлы. Для изменения этого параметра также следует нажать кнопку **"…"** указать путь к папке, где будет храниться база данных).
- имя файла. По умолчанию имена файлов аналогичны логическим именам. **Замечание**: Для добавления новых файлов данных или журналов транзакций используется кнопка **Добавить**, а для удаления кнопка **Удалить**.

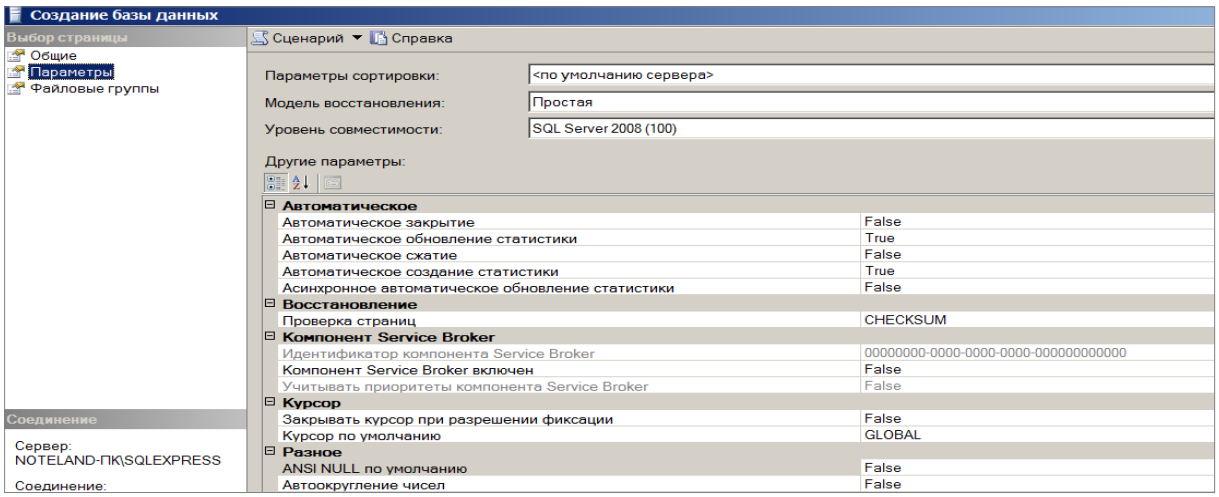


Рисунок 3

Перейдём к второстепенным настройкам файла данных. Для доступа к этим настройкам необходимо щёлкнуть мышью по пункту **Параметры.**

В правой части окна находятся следующие настройки:

- **Сортировка** – этот параметр отвечает за обработку текстовых строк, их сравнение, текстовый поиск и т.д. Рекомендуется оставить его как **по умолчанию сервера**.
- **Модель восстановления**. Данный параметр отвечает за информацию, предназначенную для восстановления БД, хранящуюся в файле транзакций. Чем полнее модель восстановления, тем больше вероятность восстановления данных при сбое системы или ошибках пользователей, но и больше размер файла журнала транзакций.
- **Уровень совместимости**, определяет совместимость файла данных с более ранними версиями сервера. Если планируется перенос данных

на другую, более раннюю версию сервера, то ее необходимо указать в этом параметре.

Рассмотрим последнюю группу настроек **Файловые группы**.

Таблица имеет следующие столбцы:

- **Имя** группы файлов.
- **Файлы** – количество файлов входящих в группу.
- **Только для чтения**. Файлы нельзя изменять.
- **По умолчанию**. Все новые файлы данных будут входить в эту группу.

Эту группу настроек оставляем без изменений.

Для принятия всех настроек и создание файла данных и журнала транзакций нашей БД в окне **Новая база данных** нажать кнопку **"Ok"**.

Произойдет возврат в окно среды разработки **"SQL Server Management Studio"**. На панели обозревателя объектов в папке **Базы данных** появится новая БД **DreamHome.**

Упражнение: просмотрите информацию о базе данных **DreamHome.**

Щёлкните правой кнопкой базу данных **DreamHome**.

Выберите в контекстном меню команду *Свойства* и просмотрите сведения о пространстве базы данных и журнала, параметры настройки (закладка *Параметры*) и параметры доступа к базе данных (закладка *Разрешения*).

Типы данных, используемые в SQL-сервере

После создания логической структуры базы данных и самой БД можно приступать к созданию в ней объектов: пользовательских типов данных и таблиц.

Одним из основных этапов в процессе создания таблиц является определение типов данных ее полей. Тип данных поля таблицы определяет тип информации, которая будет размещаться в этом поле. SQL-сервер поддерживает большое число различных типов данных: текстовые, числовые, двоичные и т.д. (см. Приложение 3).

Пользовательский тип данных

В системе SQL-сервер имеется поддержка пользовательских типов данных. Они могут использоваться при определении или какого-либо специфического формата, или наоборот, часто употребляемого. Например, когда в нескольких таблицах базы данных имеется поле с одинаковыми свойствами (типом, размером или ограничениями на значение).

Создание пользовательских типов данных в системе SQL-сервер может осуществляться следующими способами[1]:

Первый способ: При помощи процедуры *sp_addtype*

[1] Перед созданием пользовательских типов данных, выполните п.1 в разделе Порядок выполнения работы.

Синтаксис:

sp_addtype тип, системный_тип_данных [, ‘NULL’| ‘NOT NULL’]

Упражнение: С помощью процедуры *sp_addtype* создайте пользовательские типы данных для **Почтового индекса (Postcode)** и **Номера отделения (объекта, владельца и т.д.)** в базе данных DreamHome. Типы данных указаны в таблице:

Имя пользовательского типа	Стандартный тип данных	Описание данных
Postcode	char	Строка из 6 символов, допускающая значения NULL
Member_no	smallint	Целое число, не превышающее 30000

Введите и выполните следующие операторы:

EXEC sp_addtype postcode, ‘char (6)’, NULL

EXEC sp_addtype member_no, ‘smallint’

Удаление пользовательского типа данных осуществляется при помощи процедуры *sp_droptype*. Если на пользовательский тип ссылаются таблицы или объекты базы данных, его нельзя удалить.

Второй способ: С помощью обозревателя MS SQL Server Management Studio.

Для этого следует:

1. Выбрать в базе данных **DreamHome** группу ***Программирование/Типы/Определяемые пользователем типы данных.*** В результате выполнения этой операции на экран будет выведено диалоговое окно настройки нового типа данных.

2. В поле *Имя* этого диалогового окна следует указать его имя; выпадающий список *Тип данных* определяет стандартный тип данных, относительно которого будет создан пользовательский. Размерность нового типа данных указывается в поле *Точность*, а проверку наличия NULL-значений можно определить, воспользовавшись флажком *Разрешить значения NULL*.

Теперь при указании типа данных поля таблицы в списке стандартных типов данных будет присутствовать вновь созданный пользовательский тип, при выборе которого будут установлены описанные выше параметры.

При создании пользовательских типов данных следует придерживаться следующих правил:

1. Если длина поля варьируется, использовать типы данных переменной длины. Например, список имён лучше хранить в поле с типом *varchar,* чем *char* (имеющим фиксированную длину). Типы данных переменной длины помогают экономить дисковое пространство.

2. Целые и денежные типы данных, а также типы даты и времени поддерживают разные диапазоны значений в зависимости от объёма отведённой им памяти. Например, если для хранения уникальных идентификаторов объектов недвижимости используется тип *tinyint,* число объектов ограничивается 255.

3. Выбор числовых типов данных зависит от требуемой точности; обычно используется тип *decimal.*

4. Если объём информации превышает 8000 байт, следует использовать типы *text* или *image.* В противном случае – *binary, char* или *varchar. T*ипы *char* или *varchar* более функциональны по сравнению с *text* и *image.*

КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Перечислите основные способы создания базы данных в SQL Server.
2. В каких случаях создаются secondary файлы? Какой тип они имеют?
3. Какая команда служит для удаления базы данных?
4. Кому принадлежит право на удаление базы данных?
5. Какие ограничения существуют на удаление базы данных?
6. Чем отличаются типы данных Char и Varchar?
7. В каких случаях используются пользовательские типы данных?
8. Как создать пользовательский тип данных в SQL Server?

ПОРЯДОК ВЫПОЛНЕНИЯ РАБОТЫ

1. Создайте с помощью команды sp_addtype пользовательский тип данных для **Номера телефона** (обычного, мобильного, пейджера, с междугородным кодом):

Имя пользовательского типа	**Стандартный тип данных**	**Описание данных**
Phonenumber	char	Строка из 17 символов

2. Создайте с помощью настроек MS SQL Server Management Studio пользовательский тип данных:

Имя пользовательского типа	**Стандартный тип данных**	**Length**
shortstring	varchar	20

3. Создайте базу данных в соответствии с вариантом индивидуального задания.

ЛАБОРАТОРНАЯ РАБОТА № 2
СОЗДАНИЕ ОБЪЕКТОВ БАЗЫ ДАННЫХ: ТАБЛИЦ, ИНДЕКСОВ И ПЕРВИЧНЫХ КЛЮЧЕЙ ТАБЛИЦЫ. ИСПОЛЬЗОВАНИЕ ОГРАНИЧЕНИЙ

Цель работы: Освоить основные методы и приемы создания таблиц базы данных, редактирования структуры таблиц.

ОСНОВНЫЕ СВЕДЕНИЯ

Таблицы баз данных

Создание, изменение структуры и удаление таблиц

В реляционных базах данных для хранения информации используются таблицы, представляющие собой подобие двумерных массивов. Создание таблицы базы данных в системе SQL-сервер может осуществляться следующими способами:

Первый способ: С помощью SQL-команды.

Для создания таблиц в SQL Server в первую очередь необходимо сделать активной ту БД, в которой создается таблица. Для этого в новом запросе можно набрать команду: **USE** <Имя БД>, либо на панели инструментов необходимо выбрать в выпадающем списке рабочую БД. После выбора БД можно создавать таблицы.

При создании таблицы необходимо указать ее имя, имена полей и их типы данных. Для каждого поля требуется указать тип данных и, при необходимости, другие параметры.

При определении таблицы можно разрешить или запретить использование в поле значений *NULL*. Если не указано, может ли поле содержать пустые значения, SQL Server определит это самостоятельно, в зависимости от стандартных параметров для текущей базы данных.

Синтаксис:

CREATE TABLE *имя_таблицы*
(имя_поля тип_данных [NULL| NOT NULL]
[, …n])

Порядок расположения полей в таблице определяется тем, в какой последовательности они указаны в команде создания таблицы. Для создания полей, содержащих сгенерированные системой последовательные числовые значения, идентифицирующие каждую строку таблицы, можно использовать *свойство IDENTITY* (такие поля называются полями IDENTITY). Поле IDENTITY обычно служит первичным ключом.

Синтаксис:

CREATE TABLE *имя_таблицы*
(имя_поля, числовой_тип_данных

IDENTITY [(*начальное_значение, шаг*)] [NOT NULL]

Используя свойство IDENTITY,необходимо иметь в виду следующее:

- в таблице может быть только одно поле IDENTITY;
- значение поля IDENTITY нельзя изменить;
- значения NULL в поле IDENTITY не допускаются;
- в поле IDENTITY должны использоваться целые (*int, smallint* или *tinyint)*, числовые или десятичные типы данных, причём два последних надо задать с нулевой разрядностью. Выбирая тип поля, надо оценить возможное число записей таблицы;
- информацию об определении поля IDENTITY задают две системные функции: IDENT_SEED (начальное значение) и IDENT_INCR (шаг);
- получить значение ключа последней вставленной во время текущей сессии записи можно средствами функции @@IDENTITY.

Упражнение: При помощи оператора CREATE TABLE создайте таблицу OWNER в базе данных DreamHome_db, определив её поля, как указано в таблице:

Имя поля	**Тип данных**	**Значения NULL**	**Свойство IDENTITY**
Owner_no	*member_no*	Не допускаются	Seed=1 Increment=1
FName	*shortstring*	Не допускаются	Нет
LName	*shortstring*	Не допускаются	Нет
City	*shortstring*	Не допускаются	Нет
Street	*shortstring*	Не допускаются	Нет
House	*nchar*	Не допускаются	Нет
Flat	*smallint*	Допускаются	Нет
Otel_no	*phononumber*	Допускаются	Нет

```
CREATE TABLE OWNER
(Owner_no member_no IDENTITY(1,1)  NOT NULL,
FName shortstring NOT NULL,
LName shortstring NOT NULL,
City shortstring NOT NULL,
Street shortstring NOT NULL,
House nchar(6) NOT NULL,
Flat smallint NULL,
Otel_no phonenumber NULL)
```

Структуру таблицы можно изменить. Для этого используется оператор ALTER TABLE. Чаще всего с его помощью добавляют поля к таблице или изменяют их тип данных. Однако следует помнить, что новое поле станет последним по порядку в списке.

Синтаксис:

ALTER TABLE *таблица*

ADD *имя_поля тип_данных* [NULL| NOT NULL] [, …n]

ALTER TABLE *таблица*

ALTER COLUMN *имя_поля новый_тип_данных* [NULL| NOT NULL]

Для того чтобы иметь возможность удалить таблицу, пользователь должен быть ее собственником. Кроме того, перед удалением, SQL потребует очистки таблицы от данных, что позволяет избежать случайной и невосполнимой потери информации.

Синтаксис:

DROP TABLE *имя_таблицы*

После выполнения этой команды, имя таблицы больше не распознается. Перед удалением необходимо убедиться в том, что эта таблица не ссылается на другую таблицу или она не используется в каком-либо представлении.

Второй способ: С помощью конструктора таблиц SQL Server Management Studio.

Для создания таблицы необходимо:

1. Выбрать в списке объектов созданной базы данных группу *Таблицы*, открыть ее контекстное меню и выполнить команду *Создать таблицу*, после чего на экране отобразиться окно конструктора таблиц.

2. В колонку *Имя столбца* необходимо ввести название столбца таблицы, после чего определить его тип данных, воспользовавшись колонкой **Тип данных** окна дизайнера. Здесь в выпадающем списке отображается перечень всех доступных типов данных, определенных в SQL-сервере. В зависимости от типа данных система определит доступ к свойствам этих столбцов, значения которых задаются в нижней части окна *Свойства столбцов*. В СУБД имеется поддержка *NULL* значений. С помощью SQL-сервера можно определить их использование в таблицах. Убрав флажок в колонке *Разрешить значения Null* для некоторого поля, можно потребовать обязательный ввод значений в это поле.

При создании таблицы можно определить свойство *IDENTITY* для какого-либо ее поля. Для этого в первую очередь надо убрать флажок *Разрешить значения Null,* чтобы избежать неопределенности информации. Следующим шагом будет установка значения *да* в строке *Спецификация идентификаторов,* после чего требуется ввести *Начальное значение идентификатора* и *Шаг приращения идентификатора*.

3. После создания таблицы ее надо сохранить., расположенной на панели инструментов или подтвердить сохранение при закрытии конструктора таблиц.

При необходимости можно внести изменения в структуру таблицы после ее создания. Для этого надо вызвать конструктор таблиц, воспользовавшись командой *Проект* контекстного меню таблицы.

Если необходимо в процессе работы с SQL-сервером переименовать ранее созданную таблицу, то следует выбрать команду *Переименовать* контекстного меню таблицы.

Каждая таблица в SQL-сервере обладает рядом свойств, для просмотра которых необходимо воспользоваться командой *Свойства* контекстного меню таблицы.

Для удаления таблицы из базы данных SQL-сервера необходимо сначала выбрать ее в списке, после чего выполнить команду *Удалить* контекстного меню. Воспользовавшись кнопкой *Показать зависимости*, можно просмотреть перечень таблиц, связанных с данной таблицей.

Упражнение: Создайте таблицу **BRANCH** в базе данных DreamHome_db при помощи конструктора таблиц.

Выберите ***Таблицы*** вашей базы данных и в контекстном меню выберите команду ***Создать таблицу.***

Определите поля в соответствии со следующей таблицей (каждая строка соответствует одному полю):

Имя поля	Тип данных	Значения NULL	Флажок Allow Nulls
Branch_no	*member_no*	Не допускаются	Не установлен
Postcode	*postcode*	Допускаются	Установлен
City	*shortstring*	Не допускаются	Не установлен
Street	*shortstring*	Не допускаются	Не установлен
House	*nchar(10)*	Не допускаются	Не установлен
Btel_no	*phononumber*	Не допускаются	Не установлен
Fax_no	*phononumber*	Допускаются	Установлен

- Нажмите кнопку сохранения таблицы, введите имя таблицы **BRANCH** в окно сохранения таблицы. Закройте окно конструктора таблиц**.**

Создание индексов и первичных ключей таблицы

Создание ключей в системе SQL-сервер

Одним из основных понятий баз данных, используемых при контроле целостности информации, является ключ. Разделяют первичные и внешние ключи. ***Первичный ключ*** (PRIMARY KEY) – это уникальное поле (или несколько полей), однозначно определяющее записи таблицы базы данных. ***Внешние ключи*** (FOREIGN KEY) – это поля таблицы, которые соответствуют первичным ключам из других таблиц.

Создать первичный ключ можно одним из следующих способов:

Первый способ: При создании таблицы с помощью SQL-команд.

При создании первичного ключа надо помнить, что поле не должно содержать Null – значений.

CREATE TABLE *имя_таблицы*
(имя_поля тип_данных [(*размер*)] NOT NULL PRIMARY KEY,
имя_поля тип_данных [(*размер*)][NULL| NOT NULL],
…);

Второй способ: С помощью конструктора среды SQL Server Management Studio.

Для создания ключа необходимо:

- При создании таблицы выбрать нужное поле и установить первичный ключ с использованием кнопки  панели инструментов

Если данная операция была выполнена корректно, то слева от имени поля должен появиться соответствующий значок. Удаление первичного ключа производиться аналогично его установке.

Упражнение: Установите ключ в таблице BRANCH базы данных DreamHome_db с помощью дизайнера таблиц:

Откройте список объектов таблицы, выберите таблицу BRANCH, затем пункт ***Проект***

- Выделите поле *Branch_no.*

Щёлкните по кнопке 

Закройте окно и сохраните изменения.

Создание индексов в системе SQL-сервер

Индексом называется упорядоченный список полей или групп полей в таблице. Таблицы могут иметь огромное количество записей, при этом записи не находятся в каком-либо определенном порядке, поэтому на их поиск по указанному критерию может потребоваться достаточно продолжительное время. Когда создается индекс в поле, база данных запоминает соответствующий порядок всех значений этого поля в области памяти.

Индексы могут состоять из нескольких полей, при этом первое поле является как бы главным, второе упорядочивается внутри первого, третье внутри второго и т.д.

Индексы представляют собой наборы уникальных значений для некоторой таблицы с соответствующими ссылками на данные, расположенные в самой таблице. Индексы являются удобным внутренним механизмом системы SQL-сервер, с помощью которого осуществляется доступ к данным наиболее оптимальным способом.

В индексы следует включать поля, к которым часто выдаются запросы при выполнении операций поиска. К ним относятся:

1. основные ключи;
2. внешние ключи, а также другие поля, часто используемые для соединения таблиц;
3. поля, в которых производится поиск диапазонов ключевых значений;
4. поля, к которым производится упорядоченный доступ.

Создать индекс можно несколькими способами:

Первый способ: С помощью оператора CREATE INDEX.

Синтаксис:

CREATE INDEX *имя_индекса* ON *таблица (поле[, …n])*

Таблица, для которой создается индекс, должна уже существовать и содержать имена индексируемых полей. При этом имя индекса не может быть использовано для чего-либо другого в базе данных и SQL сам решает, когда он необходим для работы и использует его автоматически.

Для создания уникальных (не содержащих повторяющихся значений) индексов используется ключевое слово UNIQUE в операторе CREATE INDEX (CREATE UNIQUE INDEX …).

Для удаления индекса используется оператор DROP INDEX.

Синтаксис:
DROP INDEX *таблица.индекс[,…n]*

Второй способ: С помощью конструктора SQL Server Management Studio.

Для создания индекса необходимо:

1. Выбрать необходимую таблицу из базы данных, для которой будет определяться индекс, и открыть список ее компонентов.
2. Выбрать пункт *Индексы* и, затем, в контекстном меню пункт *Создать индекс*. При этом на экране отобразиться диалоговое окно *Создание индекса*. Указать имя индекса, выбрать тип индекса. Нажать кнопку *Добавить* и в открывшемся окне указать столбец (столбцы таблицы), участвующие в индексе.

Ограничения

Ограничения – рекомендуемый способ обеспечения целостности данных. Ограничения гарантируют корректность вводимых значений и связей между таблицами**.**

Есть ограничения целостности, немедленно проверяемые, и есть откладываемые. Откладываемые ограничения целостности поддерживаются механизмом транзакций и триггеров. Среди немедленно проверяемых ограничений выделяют следующие типы:

- ограничения целостности атрибутов, или полей: значения по умолчанию, задание обязательности или необязательности значений (NULL), задание условий на значения полей;
- ограничения целостности сущностей, или таблиц: значение первичного ключа, выражения, которые применимы ко всей таблице;
- ограничения ссылочной целостности: задание обязательности связи, т.е. задание обязательности или необязательности значений внешних ключей во взаимосвязанных отношениях, определение влияния изменений значений родительских ключей на значения внешних ключей.
- ограничения целостности, определяемой пользователем: пользовательский тип данных;

Ограничения определяются в операторах **CREATE TABLE** и **ALTER TABLE**.

Для анализа ошибок целесообразно именовать все ограничения, особенно если таблица содержит несколько ограничений одного типа. Для именования ограничений используется ключевое слово CONSTRAINT.

Синтаксис:

CREATE TABLE *имя_таблицы*
(
{<*определение_поля*>
|<*ограничение_таблицы*>}
[,…n]
)

Где

<*определение_поля*>::={*имя_поля тип_данных*}
[[CONSTRAINT *имя_ограничения*]
{DEFAULT *константное_выражение*
|CHECK (*логическое_выражение*)
|PRIMARY KEY [CLUSTERED |NON CLUSTERED]
|UNIQUE [CLUSTERED |NON CLUSTERED]
|[FOREIGN KEY] REFERENCES *таблица_на_которую_ссылаются*
[(*поле_таблицы_на_которую_ссылаются*)]
}]
[…n]

<*ограничение_таблицы*>::=
[CONSTRAINT *имя_ограничения*]
|CHECK (*логическое_выражение*)
|PRIMARY KEY [CLUSTERED |NON CLUSTERED] (*поле* [,…n])
|UNIQUE [CLUSTERED |NON CLUSTERED] (*поле* [,…n])
|FOREIGN KEY
(*поле* [,…n])
REFERENCES *таблица,_на_которую_ссылаются*
[(*имя родительской_таблицы,_на_которую_ссылаются* [,…n])]
}

Используя ограничения FOREIGN KEY можно не указывать список полей родительского ключа, если родительский ключ имеет ограничение PRIMARY KEY. А также, задавая это ограничение, можно использовать только слово REFERENCES.

Например, при создании таблицы **BUYER** это может выглядеть так:

Branch_no member_no NOT NULL,
FOREIGN KEY(Branch_no) REFERENCES BRANCH(Branch_no)
или так:
Branch_no member_no NOT NULL REFERENCES BRANCH

В соответствии со стандартом, изменение или удаление значений родительского ключа не допускается. Это означает, что нельзя изменить или удалить данные об отделении (например, при его закрытии) из таблицы **BRANCH** до тех пор, пока в таблице **BUYER** имеются клиенты, у которых

в поле Branch_no содержится номер изменяемого или удаляемого отделения. Однако довольно часто возникает необходимость в таких изменениях. В таких случаях задаются каскадирования или ограничения действий.

При необходимости, чтобы изменить или удалить текущее ссылочное значение родительского ключа существует три возможности:

1. Запретить изменения.
2. Сделав изменения в родительском ключе, произвести изменения во внешнем ключе автоматически (каскадное изменение).
3. Сделать изменение в родительском ключе и установить внешний ключ в NULL-значение автоматически.

В пределах этих возможностей выполняются все команды модификации.

Итак, изменения в родительском ключе можно разделить на ограниченные (NO ACTION), каскадируемые (CASCADE) и пустые (NULL) изменения. Например, при изменении номера отделения, данные о новом значении поля Branch_no должны быть переданы в таблицу **BUYER.** В этом случае следует указать оператор UPDATE с каскадируемыми изменениями. То есть, при создании таблицы **BUYER** указать:

ON UPDATE CASCADE.

Упражнение: Создайте таблицу **BUYER** в базе данных DreamHome с заданием ограничений.

Запишите оператор создания таблицы BUYER, предполагая наличие следующих ограничений целостности:

Для быстрой связи с покупателем должен быть задан, по крайней мере, один из двух телефонов: рабочий или домашний.

С таблицей **BRANCH** таблица **BUYER** связана обязательной связью, потому что когда потенциальный покупатель **обращается в** одно из отделений компании, данные о нём, заносятся в базу данных. Для моделирования этой связи при создании таблицы BUYER должен быть определён внешний ключ ***Branch_no*** и значение его NOT NULL (таким образом, задаётся обязательность связи)**.**

При создании таблицы должно быть указано условие UPDATE с каскадируемым эффектом.

```
CREATE TABLE BUYER
(Buyer_no member_no NOT NULL PRIMARY KEY,
FName shortstring NOT NULL,
LName shortstring NOT NULL,
City shortstring NOT NULL,
Street shortstring NOT NULL,
House nchar(6) NOT NULL,
Flat smallint NULL,
Htel_no phonenumber NULL,
Wtel_no phonenumber NULL,
```

Prof_Rooms tinyint NOT NULL,
Max_Price money NOT NULL,
Branch_no member_no NOT NULL,
CONSTRAINT FK_BUYER_BRANCH FOREIGN KEY(Branch_no) REFERENCES BRANCH ON UPDATE CASCADE,
CHECK (Htel_no IS NOT NULL OR Wtel_no IS NOT NULL)

)

Упражнение: Создание именованного первичного ключа в таблице OWNER базы данных DreamHome_db.

Напишите и выполните оператор, добавляющий ограничение PRIMARY KEY с именем *PK_Owner* для поля ***Owner_no*** таблицы OWNER.

USE DreamHome_db
ALTER TABLE OWNER
ADD CONSTRAINT PK_Owner PRIMARY KEY NONCLUS-TERED(Owner_no)
GO

Для поддержки первичного ключа и уникальных ограничений ключа автоматически создаётся уникальный индекс.

Для создания ограничений для таблицы с помощью конструктора необходимо:

- Выбрать таблицу в списке объектов базы данных и открыть список ее компонентов;
- В контекстном меню *Ограничения*, выбрать *Создать ограничение*, после чего на экране отобразиться диалоговое окно дизайнера таблиц и откроется окно *Проверочные ограничения*;
- В поле *Выражение* ввести выражение ограничения;
- Нажатие кнопки *Добавить* приведет к созданию нового ограничения, после чего в поле *Выражение* надо ввести SQL-команду проверки вводимого значения.

Например: SQL команда (ROOMS >=1 AND ROOMS <=5) или (ROOMS BETWEEN 1 AND 5) позволяет контролировать ввод значения (целое число не менее 1 и не более 5) в поле ROOMS таблицы PROPERTY.

Упражнение: Добавьте ограничение в таблицу **BRANCH** базы данных DreamHome_db, гарантирующее, что номер телефона соответствует принятому формату номеров.

Выберите таблицу BRANCH, выберите *Ограничения,* в контекстном меню выберите Создать ограничение, в поле *Выражение* введите выражение ограничения: (Btel_no LIKE ‘8(021[2-6][0-9])[0-9][0-9]-[0-9][0-9]-[0-9][0-9]’)

Нажмите кнопку ***Закрыть***.

Вставка столбцов в таблицу

После создания таблицы при необходимости в неё могут быть добавлены столбцы

В следующем примере добавляется столбец column_b типа VARCHAR(20, допускающий NULL значения в таблицу table_a.

ALTER TABLE table_a ADD column_b VARCHAR(20) NULL

Для удаления столбца используется команда DROP.

ALTER TABLE table_a DROP column_b.

Возиожно изменение типа столбца. Для этого используется команда ALTER COLUMN.

КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Какой оператор служит для создания таблиц в базе данных?
2. С помощью какого оператора можно добавить поля к таблице или изменить их тип данных?
3. Какой оператор позволяет удалить таблицу в базе данных?
4. Охарактеризуйте понятия внешний ключ и первичный ключ.
5. Дайте определение индекса и способы создания индексов в SQL Server. В чем различие между понятиями индекс и ключ?
6. Для чего применяются ограничения?
7. Какие типы ограничений Вам известны?

ПОРЯДОК ВЫПОЛНЕНИЯ РАБОТЫ

- Для базы данных DreamHome создайте таблицы STAFF и PROPERTY с помощью запросов. Типы данных определите самостоятельно на основании данных контрольного примера, приведённого в Приложении 2. При создании таблиц определите первичные ключи (поле ***Staff_no***[2] таблицы STAFF, поле ***Property_no*** таблицы PROPERTY). При создании таблиц для таблицы STAFF задайте необязательность значения внешнего ключа ***Branch_no***, а для таблицы PROPERTY – обязательность или необязательность значений внешних ключей: ***Branch_no*** (Branch_no member_no NOT NULL)***, Staff_no*** (Staff_no member_no NULL)***, Owner_no*** (Owner_no member_no NOT NULL).
- С помощью SQL – команд задайте ограничение FOREIGN KEY для таблиц STAFF и PROPERTY. Предусмотрите каскадное изменение значения Staff_no в таблице PROPERTY при изменении значения этого поля в таблице STAFF.
- Таблицу VIEWING создайте с помощью SQL -команд. При создании таблицы VIEWING определите два внешних ключа (поля ***Property_no, Buyer_no***) и ограничение PRIMARY KEY (первичным ключом будет набор из этих двух полей).
- Добавьте ограничения на ввод значений в поле **Rooms** таблицы PROPERTY и в поле **Sex** (**М(м)** мужской, **Ж(ж)** женский) таблицы STAFF с помощью дизайнера ограничений и SQL -команд соответственно.

[2] В качестве Номера сотрудника можно использовать № паспорта, соответственно тип данных: nchar(9)

- Задайте стандартное значение (значение по умолчанию) **'T'** для поля **Ptel** таблицы PROPERTY одним из способов.
- Для полей **FName**, **Position** таблицы STAFF, создайте индекс с помощью дизайнера индексов.
- Для поля **Date_View** таблицы VIEWING создайте индекс с помощью SQL – запроса.
- Просмотрите свойства созданных таблиц.

Реляционная модель данных приведена в Приложении 1.

ЛАБОРАТОРНАЯ РАБОТА № 3
ИСПОЛЬЗОВАНИЕ ДИАГРАММ ДЛЯ РАЗРАБОТКИ СТРУКТУРЫ БАЗЫ ДАННЫХ. ВВОД ДАННЫХ В ТАБЛИЦУ. МОДИФИКАЦИЯ ДАННЫХ ТАБЛИЦ

Цель работы: Освоить использование диаграмм в SQL-сервере для разработки структуры базы данных. Научиться вводить данные в таблицу различными способами. Освоить приемы и методы вставки, удаления и модификации строк.

ОСНОВНЫЕ СВЕДЕНИЯ

Использование диаграмм

Процесс разработки баз данных начинается с создания структуры данных, в которой отображаются все объекты и связи между ними. В базах данных SQL-сервера существует объект *Диаграммы баз данных*, позволяющий в графическом виде разрабатывать структуру данных. С помощью этого объекта можно создавать таблицы, определять ключи, осуществлять связи между таблицами и т.д.

Для создания диаграммы базы данных необходимо:

1. В списке объектов выбрать группу *Диаграммы баз данных*
2. В контекстном меню команду *Создать диаграмму базы данных*. Данное действие приведет к запуску мастера разработки диаграмм;
3. Выбрать необходимые таблицы в окне *Добавление таблиц*;
4. Если все действия выполнены верно, то система выведет соответствующее сообщение, после чего откроется диалоговое окно дизайнера диаграмм.

В дизайнере диаграмм существует четыре основных режима отображения таблицы:

Стандартный – просмотр параметров полей таблицы, причем имеется возможность изменения структуры таблицы;

Имена столбцов – просмотр перечня полей таблицы, причем имеется возможность установки первичных ключей;

Ключи – просмотр только ключевых полей;

Только имя – только заголовок таблицы.

Выбор данных режимов осуществляется с помощью кнопок панели инструментов или контекстного меню. Если в диаграмму необходимо добавить текстовый комментарий, следует воспользоваться кнопкой  ., после чего на экране отобразится запрос о вводе её имени. По завершении указания имени создаваемой таблицы, последняя появится на диаграмме в режиме *создания таблицы*.

Следующим этапом разработки структуры данных является создание реляционных связей с помощью внешних ключей. Для этого необходимо:

1. Выделить поле, которое является первичным ключом основной таблицы, щелкнув мышью на кнопке, расположенной слева от поля.
2. Не отпуская кнопку мыши перетащить его к полю, которое является соответствующим внешним ключом подчинённой таблицы. На экране отобразится диалоговое окно *Таблицы и столбцы*, в котором выведено имя связи, указаны таблицы первичного и внешнего ключа и имя поля связи.
3. После подтверждения создания внешнего ключа могут быть заданы дополнительные параметры настройки связи в окне *Связь по внешнему ключ*у:

Проверить существующие *данные при создании или повторном включении* – проверяет соответствие значений таблиц условиям данной связи по завершении процесса создания;

Спецификация INSERT and UPDATE – указание правил добавления и удаления: Нет действия, Каскадно, Присвоить Null, присвоить значение по умолчанию.

При сохранении созданной диаграммы структуры данных система запросит имя диаграммы и разрешение на внесение изменений в реальные объекты базы данных. Надо определить – созданная диаграмма останется только на «листе», или все изменения необходимо внести в структуру данных. Выбор кнопки *Yes* приведет к изменению структуры, после чего необходимо проверить корректность сделанных настроек.

Упражнение: *Создайте* диаграмму для базы данных DreamHome_db. В дизайнере диаграмм создайте таблицу CONTRACT базы данных DreamHome_db, определив её поля, как указано в таблице:

Имя поля	**Тип данных**	**Значения NULL**	**Флажок Allow Nulls**
Sale_no	member_no	Не допускаются	Не установлен
Notary_Office	shortstring	Не допускаются	Не установлен
Date_Contract	smalldatetime	Не допускаются	Не установлен
Service_Cost	money	Не допускаются	Не установлен
Property_no	member_no	Не допускаются	Не установлен
Buyer_no	member_no	Не допускаются	Не установлен

Выберите Диаграммы *базы данных/Создать диаграмму*.

Добавьте таблицы на диаграмму.

В диалоговом окне мастера создания диаграмм нажмите кнопку ***Готово*** для подтверждения осуществления взаимосвязи между данными в диаграмме и представленными таблицами.

Создайте таблицу CONTRACT. на панели инструментов окна дизайнера диаграмм и в появившемся запросе введите имя CONTRACT. Определите поля созданной таблицы.

Установите первичный ключ для поля ***Sale_no***.

Установите связь между таблицами PROPERTY и CONTRACT по ключевому полю ***Property_no***, а также между таблицами BUYER и CONTRACT по ключевому полю ***Buyer_no***.

Сохраните созданную диаграмму.

Диаграмма должна иметь следующий вид, приведенный в Приложении 1.

Ввод и модификация данных

Первый способ: Ввод и модификацию данных таблиц можно осуществить с помощью SQL-запросов. Значения могут быть помещены или удалены из полей таблицы тремя операторами:

INSERT – вставить;

UPDATE – модифицировать;

DELETE – удалить.

Второй способ: Для ввода и изменения содержимого таблицы необходимо выполнить следующие действия:

1. выбрать требуемую таблицу в списке;
2. открыть контекстное меню и выбрать *Изменить первые 200 строк*;

В рабочей области "Microsoft SQL Server Management Studio" проявится окно заполнения таблиц.

КОНТРОЛЬНЫЕ ВОПРОСЫ

- Для чего предназначены диаграммы в MS SQL Server?
- Как создать диаграмму?
- Можно ли в диаграмме создать новую таблицу?
- Какими способами можно ввести данные в таблицы?
- Как открыть окно для ввода данных в таблицы?

ПОРЯДОК ВЫПОЛНЕНИЯ РАБОТЫ

- Просмотрите свойства связей между таблицами в диаграмме базы данных DreamHome_db.
- Заполните таблицы BRANCH, OWNER, BUYER, STAFF, PROPERTY, VIEWING. Данные для заполнения таблиц находятся в Приложении 2.

ЛАБОРАТОРНАЯ РАБОТА № 4
СОЗДАНИЕ ЗАПРОСОВ НА ВЫБОРКУ И МОДИФИКАЦИЮ ДАННЫХ

Цель работы: Изучить способы создания межтабличных запросов к базам данных, подчиненных запросов, запросов на удаление и обновление данных.

ОСНОВНЫЕ СВЕДЕНИЯ

В SQL-сервер для манипулирования данными используется язык запросов Transact-SQL, который представляет собой версию языка SQL, переработанную компанией Microsoft.

Инструкция SELECT

Инструкция SELECT используется для отбора строк и столбцов таблиц базы данных.

Синтаксис:

SELECT [ALL|DISTINCT] *набор_атрибутов*
FROM *набор_отношений*
[WHERE *условие_отбора_строк*]
[GROUP BY *спецификация_группировки*]
[HAVING *спецификация_выбора_групп*]
[ORDER BY *спецификация_сортировки*]

Ключевое слово ALL означает, что в результирующий набор строк включаются все строки, удовлетворяющие условиям запроса, в том числе и строки-дубликаты. Ключевое слово DISTINCT означает, что в результирующий запрос включаются только различные строки.

В разделе **SELECT** атрибуты могут указываться с помощью (*). Например X.* обозначает совокупность всех атрибутов отношения X, а изолированная * – совокупность всех атрибутов всех отношений, фигурирующих в разделе FROM для создания запроса.

Таблицам могут быть присвоены имена – псевдонимы, что бывает полезно при соединении таблицы с самой собою или для доступа из вложенного подзапроса к текущей записи внешнего запроса. Псевдонимы задаются с помощью ключевого слова AS, которое может быть опущено.

Раздел **FROM** определяет таблицы или запросы, служащие источником данных. В случае если указано более одного имени таблицы, по умолчанию предполагается, что над перечисленными таблицами будет выполнена операция декартова произведения. Например, запрос

SELECT *
FROM A, B

соответствует декартову произведению отношений A и B.

Для задания типа соединения таблиц в единый набор записей, из которого будет выбираться необходимая информация, в разделе FROM используются ключевые слова JOIN и ON. Ключевое слово JOIN и его параметры указывают соединяемые таблицы и методы соединения. Ключевое слово ON указывает общие для таблиц поля.

При внутреннем соединении таблиц (INNER JOIN или JOIN) сравниваются значения общих полей этих таблиц. В окончательный набор возвращаются только те записи, которые отвечают условиям соединения.

Операция LEFT JOIN возвращает все строки из первой таблицы, соединенные с теми строками второй, для которых выполняется условие соединения.

Если во второй таблице таких строк нет, возвращаются значения NULL в строках второй таблицы. Аналогично, операция RIGHT JOIN возвращает все строки второй таблицы, соединенные с теми строками первой, для которых выполняется условие объединения.

Операции LEFT JOIN или RIGHT JOIN могут быть вложены в операцию INNER JOIN, но операция INNER JOIN не может быть вложена в операцию LEFT JOIN или RIGHT JOIN.

Раздел **WHERE** задает условия отбора строк. Имена атрибутов, входящие в предложение WHERE могут не входить в набор атрибутов, перечисленных в предложении SELECT.

В выражении условий раздела WHERE могут быть использованы следующие предикаты:

- Предикаты сравнения {=, >, <, >=, <=, <>.}.
- Предикат BETWEEN A AND B. Предикат истинен, когда сравниваемое значение попадает в заданный диапазон, включая границы диапазона.
- Предикат вхождения во множество IN (множество) истинен тогда, когда сравниваемое значение входит во множество заданных значений. При этом множество может быть задано простым перечислением или встроенным подзапросом. Одновременно существует противоположный предикат NOT IN (множество).
- Предикаты сравнения с образцом LIKE и NOT LIKE. Предикат LIKE требует задания шаблона, с которым сравнивается заданное значение.
- Предикат сравнения с неопределенным значением IS NULL. Неопределенное значение интерпретируется в реляционной модели как значение, неизвестное в данный момент времени. Это значение при появлении некоторой дополнительной информации в любой момент времени может быть заменено некоторым конкретным значением.
- Предикаты существования EXISTS и не существования NOT EXISTS.

Когда запрос включает предложение WHERE, СУБД просматривает всю таблицу по одной записи, чтобы определить является ли предикат истинным. Предикат может включать неограниченное число условий,

содержащих булевы операторы. Стандартными булевыми операторами в SQL являются *AND*, *OR* и *NOT*.

Упражнение: *С помощью SQL –команд создайте запросы вывода:*

1. перечня адресов трёхкомнатных квартир, предлагаемых для продажи в Полоцке;

```
SELECT Property_no, Street, House, Flat
FROM PROPERTY
WHERE City=’Полоцк’ AND Rooms=3;
```

2. список отделений компании, которые предлагают трехкомнатные квартиры с телефонами;

```
SELECT BRANCH.Branch_no
FROM BRANCH INNER JOIN PROPERTY ON
BRANCH.Branch_no=PROPERTY.Branch_no
WHERE (PROPERTY.Rooms=3) AND (PROPERTY.Ptel=’T’);
```

Раздел **GROUP BY** используется для создания итоговых запросов. Итоговые запросы имеют одно общее свойство: в предложении SELECT таких запросов используется, по крайней мере, одна агрегатная функция (AVG, COUNT (количество непустых значений в данном столбце), SUM, MIN, MAX, FIRST (значение столбца из первой строки результирующего набора записей), LAST(значение столбца из последней строки результирующего набора записей) и др. Агрегатные функции используются подобно именам полей в операторе SELECT, но с одним исключением: они берут имя поля как аргумент. С функциями SUM и AVG могут использоваться только числовые поля. С функциями COUNT, MAX и MIN могут использоваться как числовые, так и символьные поля.

Синтаксис: GROUP BY *имя_столбца*

Имя столбца – имя любого столбца из любой из упомянутой в разделе FROM таблицы.

Если GROUP BY расположено после WHERE создаются группы из строк, выбранных после применения раздела WHERE.

При включении раздела GROUP BY в инструкцию SELECT список полей должен состоять из итоговых функций SQL и из имен столбцов, указанных в разделе GROUP BY. В раздел GROUP BY должны быть включены все атрибуты, входящие в раздел SELECT.

Итоговые функции SQL:

- AVG(поле) – выводит среднее значение поля;
- COUNT(*) – выводит количество записей в таблице;
- COUNT(поле) – выводит количество всех значений поля;
- MAX(поле) – выводит максимальное значение поля;
- MIN(поле) – выводит минимальное значение поля;

- STDEV(поле) – выводит среднеквадратичное отклонение всех значений поля;
- SUM(поле) – суммирует все значения поля;
- TOP n [Percent] – выводит n первых записей из таблицы, либо n% записей из таблицы;

В предложение GROUP BY могут быть указаны одновременно несколько столбцов. Группы при этом определяются слева направо. Предложение GROUP BY автоматически устанавливает сортировку по возрастанию (если надо по убыванию – задать в ORDER BY).

Упражнение: С помощью SQL- команд создайте итоговые запросы:

Подсчёта количества трехкомнатных квартир, предлагаемых в Витебске и Полоцке.

```
SELECT City, COUNT(*) AS Количество_квартир
FROM PROPERTY
WHERE (Rooms=3) AND ((City=’Витебск’) OR (City=’Полоцк’))
GROUP BY City;
```

Раздел **HAVING** задает условие отбора групп строк, которые включаются в таблицу, определяемую инструкцией SELECT.

Условия отбора применяется к столбцам, указанным в разделе GROUP BY, к столбцам итоговых функций или к выражениям, содержащим итоговые функции. Если некоторая группа не удовлетворяет условию отбора, она не попадает в набор записей.

Синтаксис: HAVING *условие_отбора*

Агрегатные функции могут применяться как в выражении вывода результатов строки SELECT, так и в выражении обработки сформированных групп HAVING.

Упражнение: Выведите список и номера телефонов отделений, которые предлагают более одной трехкомнатной квартиры.

```
SELECT PROPERTY.Branch_no, BRANCH. Btel_no
FROM BRANCH, PROPERTY
WHERE PROPERTY.Branch_no=BRANCH.Branch_no
AND PROPERTY.Rooms=3
GROUP BY PROPERTY.Branch_no, BRANCH. Btel_no
HAVING COUNT(*)>1;
```

Сортировка результатов запроса

В SQL представлены специальные средства, которые позволяют совершенствовать вывод запросов:

- размещение текста в выводе запроса:

SELECT *имя_поля1, ‘текст’, имя_поля2 …*

При этом все символы, в том числе и пробелы, вставляются в вывод, поэтому этот способ можно использовать для маркировки вывода вместе со вставляемыми комментариями.

- упорядочение полей вывода:

ORDER BY *имя_поля* ASC|DESC;

Если указывается несколько полей, то столбцы вывода упорядочиваются один внутри другого, при этом можно определить ASC (возрастание) или DESC (убывание).

Упражнение: Определите количество объектов, находящихся в ведении каждого из сотрудников компании с упорядочением отделений по убыванию:

```
SELECT STAFF.Branch_no, STAFF.Staff_no, Count(*) AS Count_Staff_no
FROM STAFF INNER JOIN PROPERTY ON STAFF.Staff_no = PROPERTY.Staff_no
GROUP BY STAFF.Branch_no, STAFF.Staff_no
ORDER BY STAFF.Branch_no DESC, STAFF.Staff_no;
```

Вложение запросов

Одни запросы могут управлять другими запросами. Это можно сделать, разместив один запрос внутри другого запроса. Обычно подчиненный запрос генерирует значение, которое проверяется в предикате внешнего запроса (в предложении WHERE или HAVING), определяющего верно оно или нет. Совместно с подзапросом можно использовать предикат EXISTS, который возвращает истину, если вывод подзапроса не пуст.

Часто бывает необходимо сравнивать значения в определенных столбцах со списком значений этого же столбца из другой таблицы или запроса. В подобных случаях используется ключевое слово IN (NOT IN).

Упражнение: Выведите список сотрудников, за которыми не закреплен ни один из объектов недвижимости.

```
SELECT *
FROM STAFF
WHERE Staff_no NOT IN (SELECT Staff_no   FROM PROPERTY);
```

В подзапросах допускается использование агрегатных функций, например, для вывода списка трехкомнатных квартир, цена которых превышает среднюю цену трехкомнатной квартиры, используется запрос:

```
SELECT City, Street, House,Flat
FROM PROPERTY
WHERE Rooms=3
AND Selling_Price >(SELECT AVG(Selling_Price) FROM Property
                    WHERE Rooms=3);
```

Упражнение: Выведите список владельцев собственности, чьи объекты были осмотрены в определенный день:

```
SELECT OWNER.Owner_no, FName, LName
FROM OWNER INNER JOIN PROPERTY
ON PROPERTY.Owner_no=OWNER.Owner_no
WHERE PROPERTY.Property_no=ANY(SELECT Property_no
                               FROM VIEWING
                               WHERE  Date_View='17.01.2012');
```

В таблице VIEWING будет найдена соответствующая дата и передана в предложение WHERE. После определения даты в основном запросе из таблицы PROPERTY будут отобраны записи, удовлетворяющие заданному условию.

(В данном примере предполагается, что подзапрос должен вернуть только одно значение).

Если подчиненный запрос возвращает более одного значения, использование запроса в таком виде приведет к ошибке. В тех случаях, когда подчиненный запрос возвращает более одной строки необходимо использовать следующие ключевые слова: ANY, SOME, ALL. Подчиненный запрос в этом случае должен возвращать один столбец.

ANY или SOME – возвращает TRUE, если заданное выражение является истинным для какой-нибудь из строк возвращаемой запросом.

ALL – возвращает TRUE, если заданное выражение является истинным для всех строк возвращаемой запросом.

Оператор ALL работает таким образом, что предикат является верным, если каждое значение, выбранное подзапросом, удовлетворяет условию в предикате внешнего запроса.

Упражнение: Найдите всех сотрудников, чья заработная плата выше заработной платы любого из сотрудников отделения компании под номером (Branch_no) равным 3:

```
SELECT Staff_no, FName, LName, Salary
FROM STAFF
WHERE Salary > ALL (SELECT Salary FROM STAFF
                    WHERE Branch_no=3);
```

Коррелированные (зависимые) подзапросы

Существуют запросы, которые требуют повторного вычисления подзапроса. В этих случаях результаты, возвращаемые подзапросом, зависят от значений, передаваемых внешним запросом. При этом подзапрос выполняется для каждой стоки, которая выбирается во внешнем запросе.

Пример. Вывести список сотрудников, зарплата которых выше средней зарплаты сотрудников своего отделения

```
SELECT FNAME, Branch_no, SALARY
FROM STAFF S
```

WHERE SALARY > (SELECT AVG(SALARY) FROM STAFF WHERE Staff.Branch_no=S.Branch_no)

В этом примере результат подзапроса (средняя заработная плата сотрудников отделения компании, в котором работает сотрудник) зависит от того, для какого сотрудника выполняется подзапрос. То есть, подзапрос в данном случае нельзя вычислять независимо от основного запроса. В нем используется значение S.Branch_no, которое является переменным и зависит от строки, которую SQL сервер рассматривает в таблице STAFF.

Это реализуется следующим образом. Просматривается таблица Staff, из нее берется одна очередная запись и переписывается в таблицу с именем S. Для этой записи выполняется подзапрос – рассчитывается средняя заработная плата сотрудников того отделения компании значение которого в данный момент содержится в таблице S.

В данном случае потребовалось использование псевдонима (S) так как и внешний и вложенный запрос обращаются к одной и той же таблице.

Использование оператора *EXISTS*

Оператор **EXISTS** проверяет, возвращает ли подчиненный запрос хотя бы одну строку. Для проверки противоположного значения используется предикат NOT EXISTS.

Упражнение: Выведите данные об объектах собственности из таблицы PROPERTY только в том случае, если хотя бы один из них был осмотрен покупателями, и было получено согласие на приобретение:

```
SELECT Property_no
FROM PROPERTY
WHERE   EXISTS (SELECT Property_no    FROM VIEWING
                    WHERE Comments=’согласен’);
```

Создание таблицы из набора результатов

При помощи оператора можно поместить набор результатов запроса новую таблицу. Кроме того, этот оператор позволяет создавать и заполнять новые таблицы, а также создавать временные таблицы. Запросы к временной таблице иногда оказываются проще тех, которые пришлось бы выполнять, обращаясь к нескольким таблицам или базам данных. Оператор SELECT INTO позволяет создать локальную или глобальную временную таблицу. Для локальных таблиц используются имена, начинающиеся с символа #, а для глобальных – с символа ##.

Упражнение: Создайте таблицу, содержащую объекты собственности, находящиеся в городе Полоцке.

```
SELECT *
INTO ##PROPERTY_POLOCK
FROM PROPERTY
WHERE City=’Полоцк’;
```

Запросы на модификацию данных

SQL позволяет не только создавать запросы, но и вносить изменения в данные. Для этого используются запросы на удаление, вставку и обновление данных.

Запросы на удаление

Удаление строк из таблицы можно осуществить с помощью оператора DELETE. Следует учитывать, что оператор удаляет только целые записи таблицы, а не индивидуальные значения того или иного поля.

Синтаксис:
DELETE [*таблица.**]
FROM *таблица*
WHERE *условие_отбора*

Упражнение: Удалите из таблицы OWNER все записи, относящиеся к владельцу собственности, у которого значение поля Owner_no=10:

```
DELETE
FROM OWNER
WHERE OWNER_no=10;
```

В команде удаления возможно использование вложенного запроса. Это может быть необходимо в тех случаях, когда критерий, по которому выбираются данные, базируется на другой таблице.

Запросы на добавление

Ввод и добавление записей в SQL осуществляется с помощью оператора INSERT. Существует несколько вариантов вставки данных.

Вставка записей из другой таблицы

Оператор **INSERT** добавляет записи в уже существующую таблицу, вставляя в нее набор результатов оператора **SELECT**

Синтаксис:
INSERT [INTO] *имя_таблицы*
  SELECT *список_выборки*
  FROM *список_таблиц*
  WHERE *условие_поиска*

Добавление данных в указанные поля

Наиболее употребительный вариант команды INSERT INTO предусматривает добавление записи в существующую таблицу с указанием списка полей:

Синтаксис:
INSERT INTO *имя_таблицы (поле1, поле2,…)*
VALUES (*значение_поля1, значение_поля2…*)

При этом если перечислены не все поля, то в не перечисленные поля автоматически устанавливается значение NULL.

Если задается полный список значений новой записи, форма записи становится более короткой, так как перечень заполняемых полей после имени таблицы может не задаваться. Порядок следования значений после служебного слова VALUES должен соответствовать структуре таблицы.

Синтаксис:
INSERT INTO *имя_таблицы*
VALUES (*список_значений*)
В INSERT можно использовать подзапросы.

Упражнение: Вставьте в таблицу BUYER_2 данные тех покупателей, которые приобрели объекты собственности.

```
INSERT INTO BUYER_2
SELECT Buyer_no, FName, LName
FROM BUYER
WHERE Buyer_no =ANY(SELECT  Buyer_no
                    FROM VIEWING
                     WHERE Comments =’согласен’);
```

Упражнение: Добавьте данные в таблицу VIEWING*:*

```
INSERT INTO VIEWING (Date_View, Comments, Property_no, Buyer_no)
VALUES(‘31.03.03’,’согласен’,3000,4)
```

Запросы на обновление

Запрос на обновление реализуется с помощью оператора UPDATE. Он служит для изменения значений полей на основе заданного условия отбора.

Синтаксис:
UPDATE *имя_таблицы|имя_проекции*
SET *имя_поля={выражение*|DEFAULT|NULL}[,…n]
{[FROM *таблица |соединенная_таблица*}[,…n]]
WHERE *условие_отбора*

Упражнение: Снизить цены на квартиры, в которых не установлены телефоны на 2 %:

```
UPDATE PROPERTY
SET Selling_Price= Selling_Price*0.98
WHERE Ptel_no=’-’;
```

КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Перечислите основные разделы инструкции SELECT.
2. Для чего предназначены ключевые слова ALL и DISTINCT?

3. Объясните назначение ключевых слов INNER JOIN, LEFT JOIN, RIGHT JOIN.
4. К каким полям применяется оператор LIKE?
5. Перечислите основные агрегатные функции, которые используются в предложении SELECT?
6. Какая команда предназначена для упорядочения выводы информации?
7. В каких случаях необходимы псевдонимы таблиц?
8. В чем заключаются особенности разделов HAVING и WHERE?
9. В каком случае предикат EXISTS возвращает значение ‘TRUE’?
10. Приведите примеры вложенных запросов с предикатами ANY, ALL.

ПОРЯДОК ВЫПОЛНЕНИЯ РАБОТЫ

С помощью SQL –команд создайте следующие запросы:

1. **С использованием соединения таблиц (не использовать подчиненные запросы и агрегатные функции):**

- Вывести список объектов недвижимости, не закрепленных ни за одним сотрудником.
- Вывести адреса объектов недвижимости, которые не были осмотрены покупателями.
- Вывести данные (фамилия, номер телефона) продавцов всех осмотренных покупателями объектов собственности, у которых (объектов) в поле COMMENT таблицы VIEWING установлено значение «требует ремонта».
- Вывести данные об объектах собственности (Property_no, адрес) об объектах собственности, которые были зарегистрированы в БД более трех месяцев назад и не были проданы.
- Вывести список сотрудников (Staff_no, Fname), продавших более одного объекта недвижимости.

2. Запросы с использованием агрегатных функций.

- Вывести список сотрудников (**Staff_no**, **Fname)**, продавших более одного объекта недвижимости.
- Найти номера отделений и фамилии служащих, однофамильцы которых работают в этом же отделении.
- Определить, сколько однокомнатных, двухкомнатных, … ,n-комнатных квартир продано в каждом городе (проданные квартиры содержатся в таблице **CONTRACT**).
- Вывести цены самых дорогих и самых дешевых квартир в каждом городе.
- Найти сотрудника (**Staff_no**, **Fname)**, продавшего максимальное количество объектов недвижимости в течение последних трех месяцев.

3. Запросы с использованием подчиненных запросов:

- Вывести адреса объектов недвижимости, которые не были осмотрены покупателями (использовать **NOT IN**).
- Вывести адреса объектов недвижимости, которые не были осмотрены покупателями (использовать **NOT EXISTS**).
- Вывести адреса объектов недвижимости, которые не были осмотрены покупателями (не использовать **NOT IN, NOT EXISTS**).
- Вывести список адресов квартир в каждом городе, цены которых превышают среднюю цену квартир в этом городе.
- Определить количество сотрудников компании заработная плата которых превышают среднюю заработную плату.

4. Запросы на модификацию данных

- Снизить на 10% заработную плату сотрудников, которые не продали ни одной квартиры, и заработная плата которых превышает среднюю заработную плату в своем отделении.
- Уменьшить на 10% цены самых дорогих в своих отделениях квартир.
- В локальную временную таблицу занести список сотрудников, которые не продали ни одной квартиры в течение последнего года.
- Таблица **PROPERTY**_1 служит для хранения данных о проданных объектах собственности (находятся в таблице **CONTRACT**). С помощью команды **INSERT** вставить данные об этих квартирах из таблицы **PROPERTY** в таблицу **PROPERTY_1.**
- Удалить из базы данных записи о проданных объектах недвижимости.

ЛАБОРАТОРНАЯ РАБОТА № 5
СОЗДАНИЕ ПРЕДСТАВЛЕНИЙ ПОЛЬЗОВАТЕЛЯ, ТРИГГЕРОВ, ХРАНИМЫХ ПРОЦЕДУР, ПОЛЬЗОВАТЕЛЬСКИХ ФУНКЦИЙ

Цель работы: изучить способы создания и использования представлений, триггеров, хранимых процедур и функций.

ОСНОВНЫЕ СВЕДЕНИЯ

Представления

Механизм представлений является мощным средством СУБД, позволяющим скрыть реальную структуру БД от некоторых пользователей. Реально представление является хранимым в БД запросом, отличаясь от запроса лишь тем, что при изменении данных в таблице они автоматически

изменяются и в представлении, что обеспечивает актуальное состояние данных. Представление дает возможность пользователю работать только с теми данными, которые ему нужны, кроме того, механизм представлений позволяет скрыть служебные, конфиденциальные данные.

Создание представления базы данных в системе SQL-сервер может осуществляться следующими способами:

Первый способ: С помощью SQL запроса.

Представление создается командой CREATE VIEW, после которой указывается его имя, а далее следует запрос, формирующий тело представления:

Синтаксис:

CREATE VIEW *имя_представления*
AS SELECT …

Горизонтальные представления

Горизонтальное представление позволяет ограничить доступ пользователей определенными строками из одной или нескольких таблиц. Например, создать представление, позволяющее руководителю отделения компании под номером 3 иметь доступ только к данным сотрудников своего отделения:

```
CREATE VIEW STAFF3
AS SELECT *
 FROM STAFF
WHERE STAFF.Branch_no= 3;
```

Преимущество представления по сравнению с запросами к БД заключается в том, что оно будет модифицировано автоматически всякий раз, когда таблица, лежащая в его основе, изменяется. Например, если в отделение номер 3 будет принят новый сотрудник, то он автоматически отобразиться в представлении.

Вертикальные представления

Вертикальные представления позволяют дать доступ к информации в таблице, исключив некоторые поля. Например, для того, чтобы скрыть данные о зарплате сотрудников надо отобрать в таблицу все поля, исключая поле Salary.

```
CREATE VIEW SALARY_OFF
AS SELECT Staff_no, FName, LName, DOB, City, Street, House, Flat, Stel_no, Position, Branch_no
FROM STAFF;
```

В рассмотренном примере поля представлений имеют имена, полученные непосредственно из имен полей основной таблицы. Однако иногда возникает необходимость назвать столбцы новыми именами. Это, например, может потребоваться в случае, если столбцы являются вычисляемыми и поэтому не имеющими имен.

Имена, которые необходимо присвоить полям, записываются в круглых скобках после имени таблиц. Они могут не указываться, если совпадают с именами полей запрашиваемой таблицы.

В SQL существует понятие групповых представлений, то есть таких, которые содержат предложение GROUP BY. Представления могут быть основаны сразу на нескольких базовых таблицах.

Упражнение: С помощью SQL запроса создать представление, содержащее данные об агентах, отвечающих за продажу объектов. Представление должно включать номер отделения (Branch_no), номер работника (Staff_no) и сведения о количестве объектов, за которые он отвечает:

CREATE VIEW STAFF_PROP (Branch_no, Staff_no, Properties)

AS SELECT STAFF. Branch_no, STAFF. Staff_no, COUNT(*)

FROM STAFF INNER JOIN PROPERTY ON STAFF.Staff_no= PROPERTY.Staff_no

GROUP BY STAFF. Branch_no, STAFF. Staff_no;

Одной из причин использования представлений является стремление к упрощению многотабличных запросов. После определения представления с соединением нескольких таблиц можно использовать простейшие однотабличные запросы к этому представлению. Однако при создании запросов к представлениям, созданным на основе нескольких таблиц, следует учитывать следующие ограничения:

- если столбец в представлении создается с использованием обобщающей функции, то этот столбец может указываться только в предложениях SELECT и ORDER BY тех запросов, которые обеспечивают доступ к данному представлению. Этот столбец не может использоваться в предложении WHERE, а также не может быть аргументом в обобщающей функции;
- сгруппированное представление не должно соединяться с таблицами базы данных или другими представлениями.

Представление может быть обновляемым только в следующих случаях:

- в нем не используется ключевое слово DISTINCT;
- каждый элемент в списке предложения SELECT представляет собой имя столбца, а не выражение или обобщающую функцию;
- представление должно быть построено на базе одной таблицы;
- запрос, определяющий представление не должен содержать предложений GROUP BY и HAVING.

Хранимые процедуры

Хранимые процедуры – это подпрограммы, выполнение которых происходит непосредственно на сервере баз данных. Все хранимые процедуры в базе данных находятся в специально отведенном списке **Хранимые процедуры** группы **Программирование**.

Для создания новой процедуры с помощью конструктора необходимо:

1. Выбрать команду **Создать хранимую процедуру.** В рабочей области окна сервера появится вкладка **SQLQuery1.sql**., в котором будет расположена область для ввода текста процедуры.

Вместо текста [PROCEDURE NAME] необходимо ввести имя создаваемой процедуры, после чего набрать текст ее команд.

Чтобы посмотреть информацию о хранимой процедуре необходимо выполнить команду:

EXEC SP_HELPTEXT <Имя процедуры>

2. Далее необходимо проверить работоспособность созданной процедуры.

EXEC <МЯ_ПРОЦЕДУРЫ>

Процедура может содержать переменные-параметры, принимаемые процедурой. Каждая переменная внутри хранимой процедуры описывается следующим образом:

@<имя переменной> <тип данных>

Если в процедуру передаются параметры, то они указываются после ее имени.

EXEC <имя процедуры> <имя переменной = значение>

Такой способ передачи значений параметра в терминах SQL Server называется передачей по ссылке. При этом значения могут передаваться в произвольном порядке.

Существует другой способ передачи значений параметров в процедуру, называемый передачей значений по позиции. В этом случае параметры указываются через запятую после имени, не нарушая порядка следования параметров в теле процедуры.

EXEC <имя процедуры><имя переменной1> <имя переменной2>…

Для создания процедур в MS SQL Server используется язык Transact SQL. Каждая хранимая процедура компилируется при первом выполнении. Описание процедуры совместно с планом ее работы хранится в системных таблицах БД.

Синтаксис:

CREATE PROCEDURE *имя_процедуры (@<имя перем1> <тип данных>, @<имя перем2> <тип данных>…)*

[=значение по умолчанию] [,.параметр N] [OUTPUT]

[WITH

{ RECOMPILE

|ENCRYPTION

|RECOMPILE,ENCRYPTION}]

AS *тело_процедуры*

Создается процедура с указанным именем. Ключевое слово **RECOMPILE** определяет режим компиляции. Если **RECOMPILE** задано,

то процедура будет перекомпилироваться всякий раз, когда она будет передаваться на выполнение. Ключевое слово **ENCRYPTION** определяет режим, при котором исходный текст хранимой процедуры не сохраняется в БД.

Кроме имени все остальные параметры являются необязательными. В процедуре может быть использовано ключевое слово OUTPUT, которое определяет, что данный параметр является выходным.

Удаление процедуры осуществляется с помощью команды DROP PROCEDURE

DROP PROCEDURE [*owner*.]*procedure_name* [,[*owner*.]*procedure_name*...]

В процедурах могут использоваться следующие операторы управления:

Оператор условия

IF <*выражение*>
BEGIN
<*операторы*>
END
[ELSE]
[**IF** <*выражение*>]
BEGIN
<*операторы*>
END

WHILE <*логическое выражение*>
BEGIN
<*операторы*>
END

В этом операторе можно также использовать ключевое слово **BREAK**, которое позволяет прервать выполнение этого цикла.

Выбор одного из нескольких значений

CASE <*переменная*>
WHEN <*условие1*> **THEN** <*оператор1*>
WHEN <*условие2*> **THEN** <*оператор2*>
WHEN <*условие3*> **THEN** <*оператор3*>
…
ELSE <*оператор*>
END

Для объявления переменных, которые используются в процедуре надо воспользоваться директивой **DECLARE**. Если необходимо присвоить этой переменной какое-либо значение, используется ключевое слово **SELECT**. Оператор **PRINT** позволяет выводить текстовое сообщение на экран.

Пример:

DECLARE @X INT;

SELECT @X=@X+1;
PRINT ‘Результат’,@X;
@<Имя параметра> <Тип данных> = <Значение по умолчанию>
Параметры разделяются между собой запятыми.
Начало тела процедуры, обозначается служебным словом "BEGIN".
Тело процедуры, содержит команды языка программирования запросов T-SQL.

Упражнение: Создать процедуру для повышения заработной платы сотрудника только в том случае, если за ним закреплен хотя бы один объект собственности в таблице Property (номер сотрудника и процент повышения заработной платы передаются в процедуру как параметры).

```
CREATE PROC  NEW_SALARY
(@Staff_no int,
@Procent decimal)
AS
 IF EXISTS  (SELECT property_no
          FROM  PROPERTY
          WHERE  Staff_No= @Staff_No )
  UPDATE STAFF SET Salary=Salary*(100+   @Procent)/100
WHERE Staff_No= @STAFF_NO
```

Для запуска процедуры:

```
EXEC NEW_SALARY  @Staff_No = BMO5502601, @PROCENT=10
```

Пользовательские функции

Главным отличием пользовательских функций от хранимых процедур является то, что они всегда возвращают какой то результат. Они вызываются только при помощи оператора SELECT, аналогично встроенным функциям. Все пользовательские функции делятся на 2 вида:

Скалярные функции - функции, которые возвращают число или текст, то есть одно или несколько значений;

Табличные функции - функции, которые выводят результат в виде таблицы.

Для создания новой пользовательской функции используется команда CREATE FUNCTION имеющая следующий синтаксис:

```
CREATE FUNCTION <Имя функции>
([@<Параметр1> <Тип1>[=<Значение1>],
 @<Параметр2> <Тип2>[=<Значение2>], . . .])
RETURNS <Тип>/TABLE
AS
BEGIN
[<function ststements>]
        {RETURN < type| RETURN<Команды SQL>}
 END
```

Здесь:

Имя функции - имя создаваемой пользовательской функции.

Параметр1, Параметр2, . . - параметры передаваемые в функцию.

Значение1, Значение2, … - значения параметров по умолчанию.

Тип1, Тип2, . . . - типы данных параметров.

После служебного слова RETURNS в скалярных функциях указывается тип данных результата, который возвращает скалярная функция, либо служебное слово TABLE в табличных функциях.

После служебного слова RETURN записывается SQL команда самой функции.

После служебного слова RETURN может быть несколько команд, которые располагаются между словами BEGIN и END.

Тип данных параметра должен совпадать с типом данных выражения, в котором он используется.

Если используются несколько SQL команд и BEGIN и END, то перед END следует записать команду RETURN <результат функции>.

Упражнение: Создайте скалярную функцию для вычисления средней цены трехкомнатной квартиры в заданном как параметр отделении:

```
CREATE FUNCTION average_price
(@Branch_no Int)
RETURNS Real
AS
BEGIN
RETURN (SELECT avg(selling_price)FROM PROPERTY
     WHERE Branch_no=(@Branch_no))
END
```

Созданная функциязапускается следующим образом:

```
SELECT dbo.average_price(1).
```

Упражнение: Создайте табличную функцию для решения следующей задачи: из таблицы **STAFF** вывести поля **FName**, **LName** и столбец **Length_of_work**, который вычисляется как разница дат в годах, между датой приема на работу в компанию (Date_Joined) и текущей датой (параметр CurкDate).

```
CREATE FUNCTION length_of_work
(@CurrDate Date = GETDATE)
RETURNS TABLE
AS
RETURN (SELECT FName, LName,
     length_of_work = DATEDIFF (yy, Date_Joined, @CurrDate)
FROM STAFF)
```

Данная функция вызывается следующим образом:

```
SELECT * FROM dbo.length_of_work ('31/12/2012')
```

Триггеры

Триггер – это инструмент SQL-сервера, используемый для поддержания целостности данных в базе и выполнения бизнес-правил, слишком сложных для реализации ограничений. Триггеры – это специальный класс хранимых процедур, автоматически запускаемых при добавлении, изменении и удалении данных из таблицы. Каждый триггер привязывается к конкретной таблице. Когда пользователь пытается изменить данные в таблице, сервер автоматически запускает триггер, и только при его успешном завершении разрешается выполнение изменений.

В отличие от хранимых процедур, триггеры нельзя вызывать напрямую, в них нельзя передавать параметры. Главное их преимущество в том, что они могут содержать сложную логику обработки. С помощью триггеров осуществляются каскадные изменения данных, что позволяет сократить объем кода для обновления данных в связанных таблицах и обеспечить синхронность изменений во всех таблицах.

Триггеры могут использоваться для выдачи пользовательских сообщений об ошибках при возникновении определенных условий в процессе выполнения этого триггера. Ограничения, правила и значения по умолчанию позволяют выводить лишь системные сообщения об ошибках.

В зависимости от выполняемых пользователем действий, приводящих к запуску триггера, они делятся на три категории:

триггеры изменения (запускаются при попытке изменения данных с помощью команды **UPDATE**);

триггеры вставки (запускаются при попытке вставки данных с помощью команды **INSERT**)*;*

триггеры удаления (запускаются при попытке удаления данных с помощью команды **DELETE**).

Триггеры могут выполняться после выполнения операции (**AFTER**), или вместо выполнения операции (**INSTEAD OF**). Эти параметры определяет то, когда выполняется код триггера – до или после модификации данных. Но в любом случае триггер выполняется прежде, чем изменения будут зафиксированы в базе данных. Важнейшей особенностью триггера **INSTEAD OF** является то, что, он предназначен для выполнения кода, предусмотренного программистом, вместо того кода, выполнение которого обусловлено запросом.

При работе с триггерами доступны две специальные таблицы: таблиц вставок (**INSERTED**) и таблица удалений (**DELETED)** со структурой идентичной структуре таблицы, с которой связан триггер. Таблицы **INSERTED** и **DELETED** заполняются строками модифицируемой таблицы. При выполнении операции **DELETE** строки, удаленные из модифицируемой таблицы помещаются в таблицу **DELETED**. При выполнении операции **INSERTED**, строки, добавленные в модифицируемую таблицу, помещаются в таблицу **INSERTED**. При выполнении операции **UPDATE** для каждой измененной

строки ее исходное значение помещается в таблицу **DELETED**, а новое значение – в таблицу **INSERTED**. Данные таблиц **INSERTED** и **DELETED** можно использовать в триггере.

Для создания триггеров используется оператор **CREATE TRIGGER**. В коде оператора указывается таблица, в которой следует создать триггер, а также операторы, включаемые в триггер. Для создания триггера следует выполнить команду **Создать триггер**, выбрав папку **Триггеры** таблицы, для которой создается триггер и ввести код триггера.

Упражнение: Создайте триггер для поддержания целостности данных – проверки наличия связанной записи в главной таблице (**BRANCH)** при вводе данных в подчиненную таблицу **(PROPERTY)**.

При вводе новых объектов собственности, каждый из объектов должен быть соотнесен с каким-либо отделением компании, то есть при вводе значения атрибута **Branch_no** в таблицу **PROPERTY** необходимо проверить наличие этого значения в поле **Branch_no** таблицы **BRANCH.** Создаваемый триггер не позволит добавить новую запись в таблицу **PROPERTY**, если значение в поле **Branch_no** не совпадает ни с одним значением в поле **Branch_no** таблицы **BRANCH.**

Для создания триггера с именем INSCHECK с помощью SQL запроса выберите таблицу **PROPERTY** в списке объектов базы данных, после чего выполните команду ***Триггеры/Создать триггер.*** После этого будет открыто окно триггера, в которое введите следующий код:

```
CREATE TRIGGER INSCHECK ON PROPERTY
FOR INSERT
AS
DECLARE @X Member_no
SELECT @X= Branch_no  FROM                    INSERTED
  IF NOT EXISTS(SELECT * FROM
                    BRANCH WHERE Branch_no=@X)
BEGIN
   ROLLBACK TRAN
  RAISERROR('ОШИБКА ЦЕЛОСТНОСТИ! ОТДЕЛЕНИЕ  ОТСУТСТВУЕТ В ТАБЛИЦЕ BRANCH',16,10)
END
```

Триггер активизируется при вставке (ключевое слово **INSERT**) новой записи. После определения переменной @X (**DECLARE @X**) ей присваивается значение поля **Branch_no** добавляемой записи. В процессе использования триггера создается временная таблица **INSERTED**, хранящая в себе добавляемые значения. С помощью оператора **SELECT** переменной @X присваивается значение поля **Branch_no** из таблицы **INSERTED**, то есть значение поля **Branch_no** вновь добавляемой записи.

Следующий шаг работы триггера – проверка наличия в поле **Branch_no** таблицы **BRANCH** значения переменной @X, то есть проверка допустимости вводимого значения. Если значение не найдено, то выполняется блок операторов, заключенных в области **BEGIN …END**. С помощью команды **ROLLBACK TRAN**, используемой при работе с транзакциями, отменяется последняя операция. Оператор **RAISERROR** осуществляет выдачу сообщения об ошибке. Значения 16 и 10 определяют уровень критичности операции.

КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Для чего используются представления?
2. Происходит ли изменение данных в представлении в случае внесения изменений базовые таблицы, на основе которых они созданы?
3. В чем заключается отличие горизонтальных и вертикальных представлений?
4. Можно ли создать представление на основании нескольких таблиц?
5. Перечислите ограничения на обновление данных в представлениях.
6. Для чего предназначены триггеры?
7. В чем заключается механизм работы триггеров?
8. Какие виды триггеров существуют?
9. Какие операторы управления могут использоваться в хранимых процедурах?
10. Как запустить хранимую процедуру на выполнение?

ПОРЯДОК ВЫПОЛНЕНИЯ РАБОТЫ

I. Создайте представления:

1. Содержащее данные о средней заработной плате отделений компании. Представление должно включать номер отделения и среднюю заработную плату. С помощью представления создать запрос, возвращающий номера отделений с максимальной и минимальной средней заработной платой

2. Содержащее данные о количестве квартир, за который отвечает каждый сотрудник. Представление должно включать номер сотрудника и количество квартир, за которые он отвечает. С помощью представления создать запрос, возвращающий фамилии сотрудников, отвечающих за 3 квартиры.

3. Содержащее данные о квартирах, которые были осмотрены более 2 раз и у которых в поле comment таблицы **Viewing** записано «требует ремонта». С помощью представления создать запрос, возвращающий фамилии и номера телефонов владельцев этих квартир.

II. Создайте хранимые процедуры

1. Для вычисления количества отделений компании, в которых среднее число объектов недвижимости, за которые отвечает сотрудник меньше двух. Результат вернуть через выходной параметр.

2. Для повышения заработной платы сотрудника на заданный, как параметр, процент при условии, что его заработная плата является минимальной в своем отделении (в процедуру передаётся номер сотрудника, процент повышения заработной платы). Вывести сообщение о результате выполнения операции. Вызвать процедуру для сотрудника с заданным Staff_no.

3. Для вывода списка сотрудников заданного отделения, которые не продали ни одной квартиры заданном интервале времени (передать в процедуру дату начала и конца интервала).

4. Для снижения цены квартиры на заданный как параметр процент, если квартира была осмотрена покупателями более двух раз, но не была куплена. Предусмотрите вывод списка квартир, цены которых были понижены (с указанием цены после снижения).

5. Для удаления из базы данных информации об определенном владельце недвижимости. Если с данных владельцем связаны записи в подчиненных таблицах, удаление должно быть отменено. Вывести сообщение о результате выполнения операции.

III. Создайте пользовательские функции

1. Скалярную, для вывода адреса самой дешевой квартиры с заданным, как параметр, количеством комнат в заданном, как параметр, городе. Вызвать функцию для определения адреса самой дешевой однокомнатной квартиры в Витебске.

2. Скалярную, для вычисления количества объектов, проданных отделением в заданном интервале времени. Вызвать функцию для определения количества объектов:

а) проданных одним из отделений за определенный период времени;

б) проданных каждым отделением за определенный период времени;

в) рейтинг отделений по числу проданных объектов в заданном интервале времени.

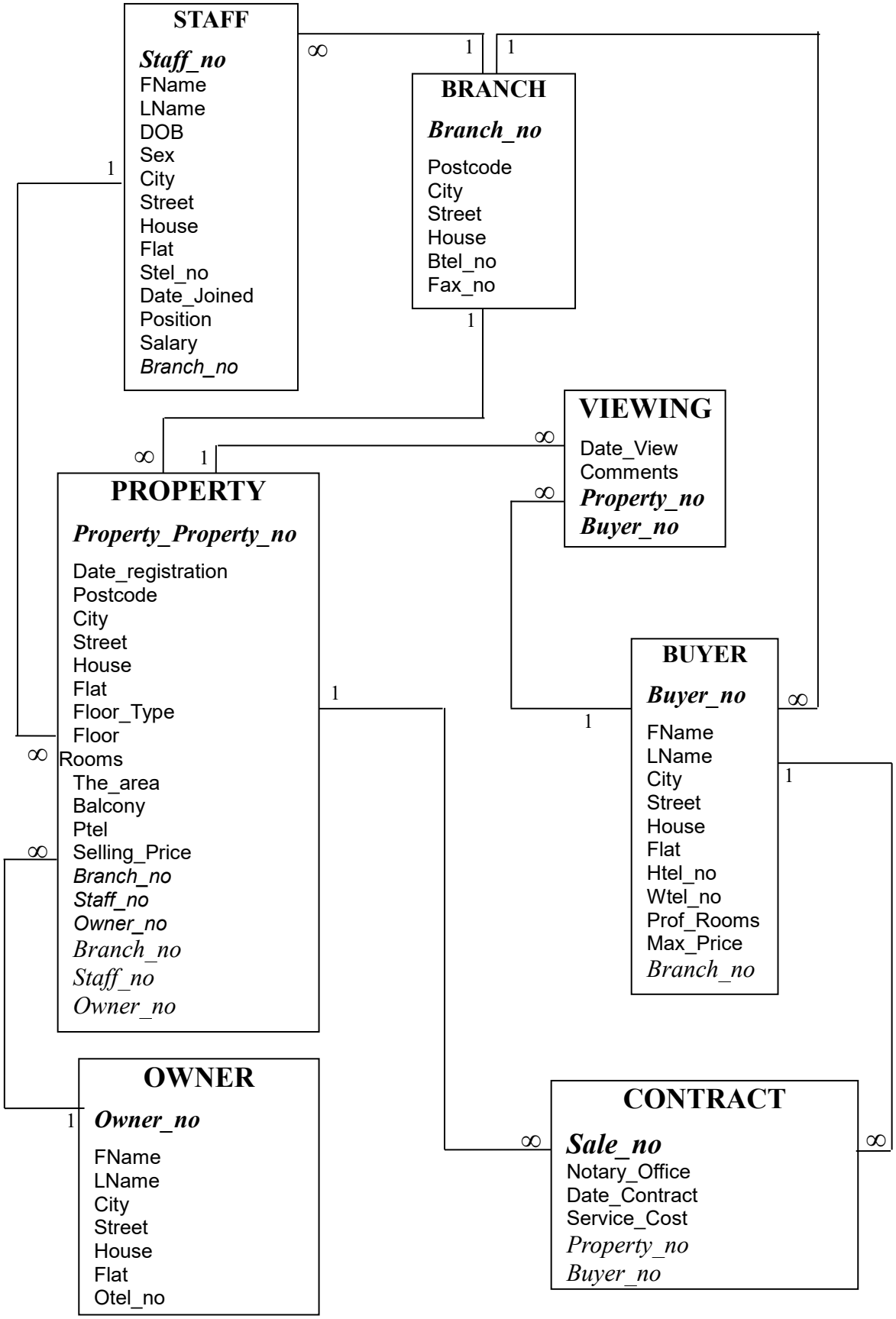
3. Табличную, возвращающую данные о среднем возрасте сотрудников отделения компании. Таблица должна включать номер отделения и средний возраст. С помощью функции создать запрос для вывода списка отделений, в которых средний возраст сотрудников превышает 40 лет,

IV. Создайте триггеры

1. для вывода сообщения о превышении количества объектов собственности, закрепленных за сотрудником, при вводе нового объекта в таблицу **PROPERTY** (количество объектов не должно быть больше трех). Проверьте работоспособность триггера.

2. для удаления данных о владельце недвижимости из таблицы **OWNER**, при продаже принадлежащего ему объекта недвижимости в том случае, если у него нет других объектов. Проверьте работоспособность триггера.

ПРИЛОЖЕНИЕ 1. РЕЛЯЦИОННАЯ МОДЕЛЬ ДАННЫХ



ПРИЛОЖЕНИЕ 2. ДАННЫЕ ДЛЯ ЗАПОЛНЕНИЯ ТАБЛИЦ

Таблица BRANCH

Branch_no	**Postcode**	City	**Street**	**House**	**Btel_no**	**Fax-no**
1	210009	Витебск	Замковая	4/офис404	8(02122)37-73-56	37-73-58
2	210033	Витебск	Суворова	9/11	8(02122)36-01-80	33-25-23
3	211440	Новополоцк	Молодёжная	18	8(02144)57-31-29	57-25-30
4	211460	Россоны	Ленина	3	8(02159)25-55-20	

Таблица Qwner

Owner_no	**FName**	**LName**	City	**Street**	**House**	**Flat**	**Otel_no**
1	Жерносек	Юрий	Витебск	Терешковой	28/1	7	62-07-94
2	Панкратова	Инна	Новополоцк	Парковая	2б	12	57-12-48
3	Амбражевич	Сергей	Новополоцк	Двинская	5	18	52-14-89
4	Поскрёбышева	Елена	Витебск	П. Бровки	26/1	40	23-00-72(аб978)
5	Титов	Николай	Витебск	Интернационалистов	35	187	8(029)633-76-68
6	Скребкова	Алла	Новополоцк	Молодёжная	1	22	8(029)688-84-46
7	Николаев	Влад	Витебск	Фрунзе	33	214	8(029)673-07-30
8	Цалко	Сергей	Лепель	Ленина	14/2	4	25-17-90
9	Цыркунова	Наталья	Россоны	Цветочная	90	15	26-32-48
10	Яковлев	Андрей	Витебск	Лазо	65		21-72-25

Таблица VIEWING

Date_ View	**Comments**	***Property_no***	***Buyer_no***
19.10.23	не согласен	3002	1
17.01.23	согласен	3003	1
21.10.23	согласен	3002	4
19.01.23	согласен	3005	2
25.03.12	требует ремонта	3001	7

Таблица BUYER

Buyer_no	**FName**	**LName**	City	**Street**	House	**Flat**	**Htel_no**	**Wtel_no**	**Prof_Rooms**	Max_Price	**Branch_no**
1	Невердасов	Виктор	Витебск	Московский пр-т	16/4	117	62-08-19	36-40-80	2	110000	2
2	Кассап	Светлана	Новопо-лоцк	Гайдара	17а	4	57-12-48		1	78000	3
3	Орлов	Александр	Минск	Либнехта	93	15	8(017)286-13-21		3	14500	1
4	Сафронова	Светлана	Витебск	пр-т Победы	3/1	324	8(029)661-07-30	22-67-94	2	60000	4
5	Окорков	Вадим	Минск	Лермонтова	35	187		8(017)224-84-24	5	300000	1
6	Семёнов	Вячеслав	Витебск	Замковая	4	13	23-00-72(аб964)		2	10500	1
7	Краснова	Жанна	Витебск	Клиническая	104			23-50-70	1	90000	2
8	Будда	Елена	Лепель	Ленина	3	5	25-17-90		4	200000	3
9	Боровая	Наталья	Орша	Смоленская	12/4	26	26-32-48		2	140000	2
10	Алипов	Игорь	Витебск	Московский пр-т	22	4	21-72-25		3	150000	2

Таблица PROPERTY

Property_no	**Data_regis-tration**	**Pstcode**	**City**	**Street**	**House**	**FFlat**	**Floor_Type**	**Floor**	**Rooms**	**The_area**	**Balcony**	**Ptel**	**Selling_Price**	**Branch_no**	**Staff_no**	**Owner_no**
3000	16.11.23	210033	Витебск	Смоленская	11	57	5П	3	1	31/18/10	Б	Т	60000	1	ВМО550262	1
3001	17.12.22	210035	Витебск	Бровки	11	49	9К	9	1	37/21/7	Бз	-	70000	2	ВМО550262	5
3002	18.10.23	210029	Витебск	Строителей	23/2	214	12П	2	2	81/29/9	Лз	Т	92000	2	ВМО550266	7
3003	19.12.22	210005	Витебск	Лазо	11	14	3К	3	2	65/40/7,6	2Бз	-	15000	1	ВМО550261	5
3004	15.10.23	211460	Россоны	Ленина	32	65	5П	5	3	68,4/44,3/9,81	2Бз	Т	100000	4	ВМО550268	7
3005	11. 09.22	211440	Новополоцк	Школьная	11	56	9П	3	1	36/18/8,2	Б	Т	75000	3	ВМО550260	6
3006	14.10.23	211440	Новополоцк	Молодёжная	5	14	9П	2	2	46/27/6,8	Лз	Т	60000	3	ВМО550264	3
3007	20.03.23	211180	Полоцк	Вокзальная	8	15	5К	5	3	65/38/7	Б	Т	80000	3	ВМО550267	2
3008	16.01.23	211460	Россоны	Советская	17	1	5К	1	3	65/38/7	-	Т	47500	4	ВМО550268	7

Таблица STAFF

Staff_no	**FName**	**LName**	DOB	Sex	City	**Street**	**House**	**Flat**	**Stel_no**	**Date_Joined**	**Position**	**Salary**	**Branch_no**
BMO550260	Батуркин	Александр	17.10.68	М	Новополоцк	Парковая	5	13	23-79-77	1.01.2011	менеджер	2500000	3
BMO550261	Чубаро	Наталья	25.05.72	Ж	Витебск	Чкалова	21/1	12	8(029)662-47-32	10.02.2022	торговый агент	1800000	1
BMO550262	Коваленко	Светлана	1.02.70	Ж	Витебск	Чкалова	24	49	62-51-23	15.01.2022	менеджер	2500000	3
BMO550263	Логинов	Вадим	9.09.67	М	Витебск	Чкалова	41/2	96	23-06-73	2.09.2019	директор	3000000	1
BMO550264	Суворов	Виталий	11.07.65	М	Новополоцк	Школьная	47/1	157	22-29-03	1.02.2020	торговый агент	1800000	3
BMO550265	Жарков	Герман	6.03.68	М	Витебск	Гагарина	31	28	63-43-98	2.10.2017	менеджер	2800000	2
BMO550266	Ганущенко	Галина	5.11.69	Ж	Витебск	Лазо	90/1	54	62-43-64	3.11.2015	курьер	1800000	2
BMO550267	Сотникова	Ольга	7.12.67	Ж	Полоцк	Вокзальная	7			2.02.2012	маклер	1000000	3
BMO550268	Янчиленко	Сергей	28.02.70	М	Россоны	Ленина	28/2	25		17.05.2017	торговый агент	1500000	4

ПРИЛОЖЕНИЕ 3. ПЕРЕЧЕНЬ ФУНКЦИЙ SQL SERVER

Функция	***Описание***
*ABS(*числовое_выражение*)*	Модуль числа
*ACOS(*вещественное_выражение*)*	Арккосинус (в радианах)
*ASCII(*символьное_выражение*)*	ASCII-код первого символа в строке
*ASIN(*вещественное_выражение*)*	Арксинус (в радианах)
*ATAN(*вещественное_выражение*)*	Арктангенс (в радианах)
CASE	Вычисление результата по списку выражений
CAST	Преобразование типа данных
CEILING	Наименьшее целое, большее либо равное заданной величине
*CHAR(*целое_выражение*)*	Преобразование целого числа в ASCII-символ
*CHARINDEX(*символьное_выражение, выражение,начало*)*	*Начальная позиция подстроки в выражении (0, если подстрока не найдена)*
*COALESCE(*выражение1, выражение2*)*	*Первое выражение, отличное от NULL*
*CONVERT(*тип_данных,выражение [,стиль]*)*	*Преобразование типа данных*
*COS(*вещественное_выражение*)*	*Косинус угла*
*COT(*вещественное_выражение*)*	*Контангенс угла*
CURRENT_USER	*Эквивалент USER_NAME( )*
*DATALENGTH(*символьное_выражение*)*	*Целое число символов в выражении, не считая завершающих пробелов*
*ROUND(*числовое_выражение, целое_выражение [,функция]*)*	*Округление числового выражения до позиции, задаваемой целым выражением. Если третий аргумент отличен от 0, ROUND не округляет, а отбрасывает цифры*

Функция	***Описание***
*DATENAME(*компонент, выражение_дата*)*	Компонент даты, выраженный в виде строки. По возможности преобразуется в имя (например, June)
*DATEPART(*компонент, выражение_дата*)*	Заданный компонент даты в виде целого числа
*DATEDIFF(*компонент, выражение_дата1, выражение_дата2*)*	Разность дат, выраженная в заданных компонентах
*DAY(*дата*)*	Целое число, определяющее день месяца
*EXP(*вещественное_выражение)	Экспонента заданного числа
*FLOOR(*числовое_выражение*)*	Наибольшее целое, меньшее либо равное заданной величине
GETDATE( )	Текущие системная дата и время
GETANSINULL(имя БД)	Выводит допустимо или недопустимо использовать в БД значение NULL
*IDENT*_SEED(таблица)	Выводит начальное значение счетчиков в таблице
*IDENT_INCR({*имя_*таблицы имя*_представления*})*	Величина приращения столбца счётчика
*IS_MEMBER({*группа | роль*})*	1, если текущий пользователь является членом группы NT или роли SQL Server, 0 – если не является и NULL, если группа/роль не определена
*IS_SRVROLEMEMBER(*роль*{,*имя*})*	1, если имя пользователя входит в роль SQL Server
*ISDATE(*выражение*)*	1, если выражение определяет допустимую дату
*ISNUMERIC(*выражение*)*	1, если выражение является числовым
*LEFT(*символьное_выражение, целое выражение*)*	Заданное число символов от начала строки
*LEN(*символьное_выражение*)*	Длина строки в символах без учета завершающих пробелов
*LOG10(*вещественное_выражение*)*	Десятичный логарифм

Функция	*Описание*
*LTRIM(*символьное_выражение*)*	Удаление начальных пробелов
*LOWER(*символьное_выражение*)*	Преобразование строки в нижний регистр
*MONTH(*дата*)*	Целый номер месяца
PI( )	Константа 3,1415926...
*POWER(*основание, степень*)*	Возведение в степень
RADIANS( )	Преобразование градусов в радианы
*RAND([*целое_выражение*])*	Получение случайного вещественного числа в интервале от 0 до 1 (необязательное целое выражение используется для раскрутки генератора случайных чисел)
*REPLACE(*символьное_выражение1, символьное_выражение2, символьное_выражение3*)*	Замена всех экземпляров строки 2 в строке 1 на строку 3
*REVERSE(*символьное_выражение*)*	Обратная перестановка строки
*RIGHT(*символьное_выражение, целое выражение*)*	Заданное число символов в конце строки
*ROUND(*числовое_выражение, целое_выражение *[,* функция*])*	Округление числового выражения до позиции, задаваемой целым выражением. Если третий аргумент отличен от 0, ROUND не округляет, а отбрасывает цифры
*RTRIM(*символьное_выражение*)*	Удаление завершающих пробелов
*SIGN(*целое_выражение*)*	+1 для положительных чисел, -1 для отрицательных и 0 для нуля
*SIN(*вещественное_выражение*)*	Синус угла
*SPACE(*целое_выражение*)*	Построение строки из заданного количества пробелов
*SQRT(*вещественное_выражение*)*	Извлечение квадратного корня
*SQUARE(*вещественное_выражение*)*	Возведение в квадрат
*STR(*вещественное_выражение *[,*длина *[,* точность*]])*	Преобразование числа в строку
*STUFF(*символьное_выражение, начало, длина, символьное_ выражение2*)*	n символов строки 1, начиная с заданной начальной позиции, заменяются строкой 2

Функция	*Описание*
*SUBSTRING(*выражение, начало, длина*)*	Получение подстроки
*SUSER_ID([*имя_пользователя*])*	Идентификатор пользователя SQL Server
SYSTEM_USER	Имя текущего пользователя SQL Server
*TAN(*вещественное_выражение*)*	Тангенс угла
*UPPER(*символьное_выражение*)*	Преобразование строки в верхний регистр
*USER_ID(*имя_в_базе*)*	Идентификатор пользователя в базе данных
*USER_NAME([*идентификатор_пользователя*]) или SESSION_USER*	Имя пользователя в базе данных
*YEAR(*выражение_дата*)*	Год в виде четырёх цифр
SUM([all | distinct] выражение*)*	Вычисляет общее количество (разных) значений в числовом столбце
AVG([all | distinct] выражение*)*	Вычисляет среднее арифметическое (разных)
COUNT([all | distinct] выражение*)*	Вычисляет количество (разных) значений в числовом столбце, отличных от NULL
COUNT(*)	Вычисляет количество выбранных записей. Единственная агрегатная функция, учитывающая значения NULL
*MAX (*выражение*)*	Определяет максимум среди выбранных значений
*MIN(*выражение*)*	Определяет минимум среди выбранных значе-
*STDEV(*выражение*)*	Вычисляет среднее квадратичное отклонение
*STDEVP(*выражение*)*	Вычисляет среднее квадратичное отклонение для выборки
*VAR(*выражение*)*	Вычисляет статистическую дисперсию
*VARP(*выражение*)*	Вычисляет статистическую дисперсию

Учебное издание

АДАМЕНКО Наталья Дмитриевна

БАЗЫ ДАННЫХ

Методические рекомендации
к выполнению лабораторных работ

Технический редактор *Г.В. Разбоева*
Компьютерный дизайн *Л.В. Рудницкая*

Подписано в печать 20.12.2023. Формат 60x84 $^{1}/_{16}$. Бумага офсетная.
Усл. печ. л. 3,20. Уч.-изд. л. 2,32. Тираж 9 экз. Заказ 156.

Издатель и полиграфическое исполнение – учреждение образования
«Витебский государственный университет имени П.М. Машерова».

Свидетельство о государственной регистрации в качестве издателя,
изготовителя, распространителя печатных изданий
№ 1/255 от 31.03.2014.

Отпечатано на ризографе учреждения образования
«Витебский государственный университет имени П.М. Машерова».
210038, г. Витебск, Московский проспект, 33.