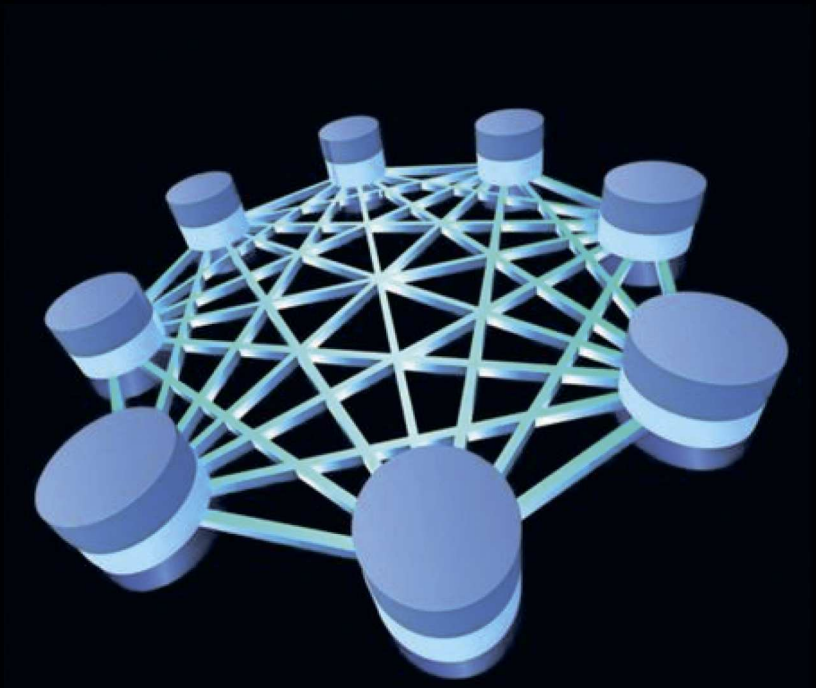
А. И. Митин

РАБОТА С БАЗАМИ ДАННЫХ MICROSOFT SQL SERVER

Сценарии практических занятий



DirectMEDIA

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ ПСИХОЛОГО-ПЕДАГОГИЧЕСКИЙ
УНИВЕРСИТЕТ»
ФАКУЛЬТЕТ «ИНФОРМАЦИОННЫЕ ТЕХНОЛОГИИ»
КАФЕДРА «ПРИКЛАДНАЯ ИНФОРМАТИКА
И МУЛЬТИМЕДИЙНЫЕ ТЕХНОЛОГИИ»

А. И. Митин

РАБОТА С БАЗАМИ ДАННЫХ MICROSOFT SQL SERVER

Сценарии практических занятий

Рекомендовано к изданию
кафедрой «Прикладная информатика
и мультимедийные технологии»

DirectMEDIA

Москва
Берлин
2020

УДК 075.8:002
ББК 73.6
М66

Рецензент:
Кулик С. Д. — доктор технических наук, профессор

Митин, А. И.

М66 Работа с базами данных Microsoft SQL Server: сценарии практических занятий / А. И. Митин. — Москва ; Берлин : Директ-Медиа, 2020. — 142 с.

ISBN 978-5-4499-0420-1

В пособии рассматриваются технологические приемы, позволяющие выполнить ряд практических заданий по работе с базами данных в среде популярной СУБД Microsoft SQL Server. Пособие может быть рекомендовано в качестве методического руководства на семинарских занятиях и при подготовке к сдаче зачетов и экзаменов по курсам, связанным с базами данных.

Текст приводится в авторской редакции.

УДК 075.8:002
ББК 73.6

ISBN 978-5-4499-0420-1

Содержание

ВВЕДЕНИЕ

Настоящее пособие описывает технологии работы с популярной системой управления базами данных Microsoft SQL Server, широко использующейся для поддержки читающихся в МГППУ курсов по разработке и эксплуатации баз данных и информационных систем.

Актуальность пособия обусловлена не только сравнительной новизной и сложностью технологии «клиент – сервер», лежащей в основе Microsoft SQL Server, но и необходимостью систематического изложения особенностей работы с нею с определенным методическим подтекстом, важным с точки зрения автора. Конечно, в наше время несложно получить информацию о технологиях работы с любыми базами данных (как в виде опубликованного пособия, так и в Интернете). Однако в основу настоящего пособия положен именно *сценарный подход* (как указано в подзаголовке), что делает пособие не просто источником информации, а своеобразным «руководством к действию» при решении практических задач. Кроме того, автор при написании пособия часто обращал внимание на методику использования апробированных на практических занятиях *технологических нюансов* Microsoft SQL Server, без которых работа с СУБД теряла бы свою эффективность.

Структура пособия соответствует тематическому плану дисциплины «Основы разработки информационных систем и баз данных» (специальность «Математическое обеспечение и администрирование информационных систем»). Материал пособия может оказаться полезным и при изучении других курсов этой специальности и других специальностей, связанных с базами данных («Базы данных», «Эксплуатация информационных систем и баз данных» и т.п.).

Изложение приемов работы с Microsoft SQL Server в пособии ведется на основе достаточно автономных *упражнений*, которые студенты могут выполнять как в рамках семинарских занятий, так и самостоятельно (например, при подготовке к сдаче зачета или экзамена). Большинство упражнений представляет собой «классические» упражнения сертификационных курсов

фирмы Microsoft [14]; содержание некоторых дополнительных упражнений взято из книг [9] и [10]. Файлы данных для упражнений располагаются в папке ***SQL_Server*** файлового сервера локальной сети.

Упражнения практически не дублируют друг друга, но допускается *возврат к ранее пройденным упражнениям* с целью вспомнить особенности их выполнения или выполнить их еще раз после изучения некоторых новых возможностей Microsoft SQL Server (такие возвраты в явном виде описываются в сценариях).

Конечно, сценарный подход предполагает определенную строгость выполнения действий студентами (в этом залог успешности выполнения упражнения в целом), но элемент творчества (в разумных пределах) сценарии не исключают (иногда по этому поводу в сценариях даже даются дополнительные рекомендации и предлагаются факультативные варианты действий).

Следует иметь в виду, что для выполнения подавляющего большинства упражнений принято не использовать достаточно развитый графический интерфейс Microsoft SQL Server (Management Studio), а *вводить тексты соответствующих запросов* непосредственно на языке Transact SQL (T-SQL). Этот подход позволяет подвести единую лингвистическую и технологическую базу под достаточно разнородные процессы, описываемые в упражнениях (создание разнообразных объектов; выборка из баз данных; модификация данных; работа с представлениями, триггерами, курсорами, процедурами; управление ходом выполнения команд и т.п.). Кроме того, использование такого вербального подхода сильно упрощает работу студентов с ограниченными возможностями здоровья.

Графический интерфейс Management Studio в этом случае удобно использовать для проверки правильности выполнения запросов.

При работе с языком T-SQL следует учесть следующие моменты:

- Большого объема ввода при работе с редактором запросов можно избежать, если использовать специальные *текстовые заготовки*, которые хранятся в папке ***SQL_Server***. Практи-

чески все тексты, набранные в пособии **моноширинным** шрифтом, можно не вводить, а копировать из соответствующих файлов заготовок.

- Установленные в компьютерных классах версии Microsoft SQL Server работают в режиме нечувствительности к регистру клавиатуры. Таким образом, команды, служебные слова и идентификаторы можно вводить как заглавными, так и строчными буквами.
- Очень часто результат выполнения операторов создания или модификации объектов (баз данных, таблиц, представлений и т.п.) не виден сразу же после выполнения. Для отображения результата надо в дереве объектов обозревателя объектов Management Studio для соответствующего *типа* объектов выполнить команду ***➢Обновить*** контекстного меню (или выделить тип объектов в дереве и нажать [F5]).
- В случае, если по тем или иным причинам конкретная база данных не существует или испорчена, ее можно *восстановить* выполнением соответствующего сценария (файла с расширением *.sql*), хранящегося в подпапке ***Восстановление БД*** папки ***SQL_Server.*** Диаграммы (схемы) баз данных, использующихся в пособии, содержатся в Приложении.

Для большей лаконичности описания технологических приемов в пособии используется специализированный язык, заимствованный из книги [7]. Он позволяет быстро и однозначно описывать работу с меню или панелями инструментов с возможностью встраивания «технологических кусков» в текст на естественном языке. Синтаксис этого языка (табл. 1) ориентирован на *работу с меню* (для обучения это наиболее естественная работа). Если же возникает необходимость использовать пиктографическую кнопку панели инструментов или диалогового окна, то в тексте пособия эта кнопка будет просто изображаться.

Изложение материала ориентировано на текущую версию Microsoft SQL Server, установленную в компьютерных классах МГППУ (конкретно, на локализованный вариант Microsoft SQL Server 2005). Однако большинство сценарных действий применимо и для других версий этого программного продукта.

Таблица 1 — Язык описания технологических приемов

Используемое обозначение	*Выполняемое действие*
❒ ***Название окна*** 🗗 ***Название подокна***	Активизация окна (подокна), то есть фиксация курсора в области окна (подокна) для его активизации.
🗀 ***Название вкладки***	Выбор ярлычка вкладки, то есть фиксация курсора на указанном ярлычке.
Название типа объекта → Название объекта	Движение на один уровень вниз по дереву объектов обозревателя объектов Management Studio (как правило, с помощью кнопки ⊞).
⊙ ***Название переключателя***	Выбор переключателя с указанным названием в активном диалоговом окне, то есть фиксация курсора на указанном переключателе.
☑ ***Название флажка***	Отметка флажка с указанным названием в активном диалоговом окне, то есть фиксация курсора на указанном флажке. Для сброса флажка используется словесное указание ☑ ***Название флажка*** **(сбросить)**
➢ ***Название команды***	Выбор в текущем меню команды с указанным названием, то есть фиксация курсора на указанной команде.
❑ ***Название кнопки***	Фиксация курсора на кнопке с указанным названием на панели инструментов, в строке состояния, или активном диалоговом окне.
Изображение кнопки ***Название кнопки***	Фиксация курсора на кнопке с указанным названием на панели инструментов, в строке состояния или в активном диалоговом окне.
Изображение вкладки ***Название вкладки***	Фиксация курсора на вкладке с указанным названием на панели команд.
Изображение пиктограммы ***Название пиктограммы***	Двойная фиксация курсора на указанной пиктограмме.
⇩ ***Название раскрывающегося списка или пункта списка-меню***	Развертывание раскрывающегося списка, то есть фиксация курсора на кнопке раскрытия списка  , раскрытие списка-меню.

Окончание табл. 1

Используемое обозначение	*Выполняемое действие*
Слово-ссылка или изображение пиктограммы	Выбор гипертекстовой ссылки, то есть фиксация курсора на слове-ссылке или пиктограмме-ссылке.
Фрагмент текста, ячейка или интервал ячеек электронной таблицы, элемент списка или раскрывающегося списка	Выделение указанного фрагмента текста, интервала или элемента списка с помощью движения мыши при нажатой и удерживаемой левой кнопке либо с помощью клавиш управления курсором при нажатой и удерживаемой клавише [Shift].
Название поля := значение	Ввод значения с клавиатуры в текстовое поле ввода, раскрывающийся список или счетчик. Значение счетчика могут также изменяться с помощью кнопок регуляторов. Вводимые с клавиатуры значения выделены в настоящем пособии подчеркиванием*.
[Клавиша]	Нажатие соответствующей клавиши на клавиатуре.
[Клавиша1] + [Клавиша2]	Нажатие первой из указанных клавиш и нажатие второй клавиши при удерживаемой первой клавише.
[Клавиша1] , [Клавиша2]	Последовательное нажатие сначала первой, затем второй клавиши на клавиатуре.
{Текст примечания}	Примечание к данному выполняемому действию или к параметрам этого действия.

* В некоторых случаях в качестве вводимого значения указывается не конкретный текст, а смысловое описание этого текста, вместо которого пользователь должен ввести подходящее по контексту конкретное значение. В таком случае смысловое описание заключается в угловые скобки, например, ***Файл*** := <имя файла>

§ 1. ПРАКТИЧЕСКОЕ ЗАНЯТИЕ 1
Изучение основных понятий структуры БД

Занятие начинается со знакомства с интерфейсом Management Studio и «воспоминаний» (на основе информации пройденных курсов) о нормализации БД, таблицах SQL Server, связях между ними и т.п. (упражнение *«Определение главных объектов в БД SQL Server»* (§ 1.1)).

Последующие упражнения этого весьма емкого занятия, по сути, следуют логике проектирования и разработки реальной базы данных (конкретно, базы данных книжного магазина – BookShopDB):

- формирование подходов к концептуальному проектированию БД, анализ сценария (постановки) задачи (упражнение *«Определение требований к БД. Сценарий БД (книжный магазин)»* (§ 1.2));
- логическое проектирование БД (сущности / связи / атрибуты) (упражнение *«Разработка логической модели данных»* (§ 1.3));
- создание БД, ее просмотр (*всех* объектов) и удаление (упражнение *«Создание и управление базой данных»* (§ 1.4));
- определение типов данных для всех столбцов таблиц (упражнение *«Определение типов данных для столбцов»* (§ 1.5));
- создание, модификация и удаление таблиц (упражнение *«Создание таблиц и управление ими»* (§ 1.6));
- учет соображений по видам целостности данных (конструкции DEFAULT, IDENTITY, NOT NULL) (упражнение *«Определение свойств, гарантирующих целостность данных»* (§ 1.7));
- добавление первичных и внешних ключей (конструкции PRIMARY KEY, FOREIGN KEY), ограничений CHECK в существующие таблицы (упражнение *«Добавление ограничений в существующие таблицы»* (§ 1.8)).

§1.1. Упражнение
«Определение главных объектов в БД SQL Server»

Общее замечание. Данное упражнение является *первым* в большом цикле упражнений этого и последующих практических занятий. Ввиду этого его цель является двойственной: не только ознакомиться с интерфейсом основной среды работы студента (Management Studio), но также получить определенную практику (или даже вспомнить материал прошлых курсов) в применении принципов нормализации БД и определении структуры БД (сущности, связи).

1. Убедиться, что ядро SQL Server работает (пиктограммы в панели задач может и не быть); если это не так, то при запуске Management Studio будет выдаваться ошибка соединения.

Для активизации ядра в меню «Пуск» выполнить запуск диспетчера конфигурации: ➢*Программы* ➢*Microsoft SQL Server 2005* ➢*Средства настройки* ➢*SQL Server Configuration Manager.*

В диалоговом окне диспетчера конфигурации выполнить:

⇗ *Службы SQL Server 2005*

⇗ *SQL Server (MSSQLSERVER)*

❑▶

Замечание. Действия п. 1, возможно, придется выполнять в начале каждого занятия, если не установить режим автоматического запуска: в контекстном меню ядра сервера в диспетчере конфигурации выполнить ➢*Свойства*, а в открывшемся диалоговом окне ☐ *Свойства: SQL Server (MSSQLSERVER)* установить:

🗀*Служба*

Режим запуска ⇩*Авто*

❑*OK*

2. Вызвать Management Studio через меню «Пуск»: ➢***Программы ➢Microsoft SQL Server 2005 ➢ SQL Server Management Studio.***

Установить связь с ядром SQL Server через Management Studio в окне соединения (☐ ***Соединение с сервером***):

Тип сервера: ⇩ *Компонент Database Engine*

Имя сервера: ⇩ *<имя сервера по умолчанию>*

Проверка подлинности:⇩ *Проверка подлинности Windows*

❑ *Соединить*

3. Просмотреть список объектов SQL Server в окне обозревателя объектов.

Найти в списке базы данных Northwind и Pubs (через *Базы данных* → *Northwind* и *Базы данных* → *Pubs*).

Замечание. Если указанные базы данных в списке отсутствуют, то следует их восстановить с помощью выполнения сценариев (файлы *InstNwnd.sql* и *InstPubs.sql* из папки ***Восстановление БД***, находящейся в указанной во введении папке ***SQL_Server***). Заметим, что выполнение сценариев требует *повторного* соединения с SQL Server. Для того, чтобы восстановленные базы данных появились в списке баз, необходимо в контекстном меню объекта ***Базы данных*** выполнить пункт ➢***Обновить.***

4. В окне обозревателя объектов открыть узел дерева ***Northwind*** и найти там пользовательские и системные таблицы, представления, хранимые процедуры, функции, пользовательские типы данных.

Обратить внимание, что перед именем любого табличного объекта указано имя владельца (у пользовательских и системных таблиц – *dbo* (data base owner)).

Просмотр содержимого таблицы

5. В контекстном меню таблицы dbo.Categories выполнить пункт ➢***Открыть таблицу.*** Разобраться с атрибутами таблицы и просмотреть строки с данными (значениями атрибутов).

Закрыть вкладку открытой таблицы dbo.Categories, выполнив пункт контекстного меню вкладки ➢***Закрыть.***

6. Ознакомиться таким же образом с несколькими другими пользовательскими и *системными* таблицами.

Просмотр табличных данных с помощью системной хранимой процедуры sys.sp_help

7. Открыть окно редактора запросов по ❑ ***Создать запрос.***

Ввести следующие операторы T-SQL:

```
USE Northwind
```

```
GO
sp_help
```

Обратить внимание на изменение цвета (разметку) конструкций **USE** и **sp_help.**

8. Выполнить запрос кнопкой ***Выполнить запрос*** и увидеть во вкладке ***Результаты*** *полный список* объектов базы данных Northwind.

Попробовать выполнить тот же запрос *перетаскиванием* мышью имени хранимой процедуры sys.sp_help из списка объектов БД Northwind в окно редактора запросов.

9. После ознакомления с объектами базы данных можно закрыть вкладку запроса выполнением пункта контекстного меню вкладки ***Закрыть.***

Нормализация структуры базы данных

Общее замечание. Эта часть практического занятия не связана впрямую с реальной таблицей Customers базы данных Northwind (хотя таблица с этим именем в БД есть) и преследует цели «напоминания» теоретических принципов нормализации БД.

10. Ознакомиться с начальным видом таблицы Customers (Покупатели):

FirstName	LastName	City
Elizabeth	Boyle	Cleveland
Rob	Caron	Chicago
Neil	Smith	Denver
Denise	Smith	Boston

Таблица содержит имя (FirstName), фамилию (LastName) и город (City) покупателя.

Выяснить соблюдение четырех основных правил нормализованности таблиц:

- в таблице должно быть поле (уникального) идентификатора;
- в таблице должна храниться информация только об одном типе объектов;

• следует избегать в таблице столбцов, допускающих пустые значения (NULL);

• в таблице не должно быть повторяющихся значений или столбцов.

11. Ознакомиться с таблицей Customers, изменяющей вид таблицы, приведенный в п. 10:

CustID	FirstName	LastName	City	PurchaseType
101	Elizabeth	Boyle	Cleveland	Books, CDs
102	Rob	Caron	Chicago	Books, videos
103	Neil	Smith	Denver	CDs, videos, DVDs
104	Denise	Smith	Boston	Books

В таблицу добавлены уникальный идентификатор покупателя (CustID) и список типов продуктов, приобретенных покупателем (PurchaseType).

Определить, какие правила нормализации нарушены и как надо изменить текущую структуру таблиц.

12. Ознакомиться с расширенной таблицей Customers:

CustID	FirstName	LastName	City	Purchase	Manufacturer	ManContact
101	Elizabeth	Boyle	Cleveland	Spring candles	Pavlova, Ltd	Ian Devling
102	Rob	Caron	Chicago	Sandal-wood incense	Mayumi's	Mayumi Ohno
103	Neil	Smith	Denver	Sage	Pavlova, Ltd	Ian Devling
104	Denise	Smith	Boston	Hanging crystal	Leka Trading	Chandra Leka

В таблицу вместо типов продуктов добавлены указатели конкретных продуктов, приобретенных покупателем (Purchase), а также данные о фирме-производителе (Manufacturer) и ее представителе (ManContact).

Определить, какие правила нормализации нарушены и как надо модифицировать структуру БД для устранения нарушений.

Создание диаграмм баз данных в SQL Server

13. Открыть узел, соответствующий базе данных Pubs, в окне обозревателя объектов.

Перейти к узлу ***Pubs → Диаграммы базы данных***.

14. В контекстном меню этого узла выполнить пункт ➢***Создать диаграмму базы данных.***

В диалоговом окне ☐***Добавление таблицы*** выделить *все* таблицы и нажать сначала ❑***Добавить***, а затем ❑***Закрыть.***

Убедиться, что все связи между таблицами строятся автоматически (вид диаграммы приведен в Приложении).

15. Можно ознакомиться с автоматическим «всплыванием» подписей связей между таблицами, подвигать таблицы с помощью мыши, переустановить масштаб диаграммы через ее контекстное меню и т.п. Можно сохранить диаграмму под каким-либо именем (при закрытии окна диаграммы, например).

Факультативно. Попробовать построить *частичную* диаграмму – например, с таблицами Authors, TitleAuthor и Titles (как иллюстрация отношений «многие к многим»).

§1.2. Упражнение «*Определение требований к БД. Сценарий БД (книжный магазин)*»

Определение целей создания системы

1. Ознакомиться со сценарием (файл *Сценарий базы данных для книжного магазина.doc*). Не нужно сразу вникать в подробности, достаточно составить представление об общих целях проекта (все равно сценарий придется перечитывать и анализировать).

Полный (в оригинале размеченный) текст сценария выглядит следующим образом:

Управляющий небольшого книжного магазина попросил вас спроектировать и реализовать базу данных для централизованного хранения информации, чтобы облегчить и сделать более эффективными управление складскими запасами, учет заказов и продаж. Магазин занимается продажей редких и букинистических книг, поэтому общее число книг в магазине, как правило, не превышает нескольких тысяч.

В настоящее время управляющий ведет всю документацию по продажам и инвентаризации на бумаге. Для каждой книги он записывает название, автора, издательство, дату публикации,

номер редакции, стоимость, рекомендованную розничную цену и оценку состояния книги. Последний параметр оценивается следующими категориями: превосходное, отличное, хорошее, неплохое, плохое, книга повреждена. Управляющему хотелось бы иметь возможность добавить краткое, длиной всего в пару предложений, описание каждой оценки, но так, чтобы это описание не было обязательным. Таким образом, сведения для любой книги состоят из ее названия, автора, стоимости, рекомендованной розничной цены и оценки состояния. Название издательства, дата выхода и номер редакции указываются не всегда. В любом случае год издания не может быть меньше 1600 и больше 2099 (последнее в соответствии с назначением новой СУБД).

Поскольку магазин работает с редкими книгами, следует учитывать каждый экземпляр по отдельности, даже если это одна и та же книга (с идентичным заглавием, автором, издательством, датой выхода и редакцией). В настоящее время управляющий назначает каждой книге уникальный идентификатор, позволяющий различать экземпляры одной и той же книги. Этот идентификатор необходимо включить в сведения о книге. Он состоит из восьми символов — цифр и букв. Управляющий также записывает краткие сведения о каждом авторе, чьи произведения продает или когда-либо продавал магазин. В магазине иногда представлены несколько книг одного автора, а одну книгу иногда пишут несколько авторов. В настоящее время у управляющего имеются сведения примерно о 2500 авторах. Сведения об авторе состоят из его имени, фамилии, года рождения и (в отдельных случаях) года смерти. Из этой информации необходимо, по крайней мере, имя автора. Управляющий также

хотел бы добавлять краткую информацию об авторе (если она есть) — одно-два предложения.

Штат магазина состоит из 12 работников (включая управляющего и его помощника). В течение ближайших нескольких лет управляющий планирует нанимать дополнительно по одному работнику в год. Как управляющему, так и его помощнику требуется возможность доступа и, при необходимости, модификации сведений о каждом работнике. В эти сведения входят имя, фамилия, адрес, номер телефона, дата рождения, дата приема на работу и название занимаемой в магазине должности. Штатное расписание магазина предусматривает должности: управляющего, помощника управляющего, продавца на полный рабочий день и продавца на неполный рабочий день. В какой-то момент у управляющего может возникнуть желание добавить к этому списку новые должности или изменить существующие, а также кратко описать обязанности для каждой должности (по крайней мере, для некоторых из них). Сотрудник может занимать только одну должность. Ни у кого, кроме управляющего и его помощника, не должно быть доступа к сведениям о персонале. Управляющему также хотелось бы вести учет числа и вида книг, продаваемых каждым работником.

В настоящее время магазин также собирает сведения о покупателях: имя, фамилия, номер телефона, почтовый адрес, купленные книги и дата покупки. Поскольку некоторым покупателям не нравится сообщать о себе личные сведения, необходимыми считаются лишь имя и фамилия. На данный момент в список управляющего занесено 2000 покупателей, большинство из которых, хотя и не все, делали в магазине покупки.

Управляющему необходимо регистрировать продажи, отслеживая заказ с момента его приема

продавцом до оплаты. Иногда, например, когда покупатель лично посетил магазин, эти события происходят одновременно. В каждом заказе необходимо указывать сведения о проданной книге, ее покупателе, оформившем покупку продавце, числе проданных экземпляров и дате заказа. А кроме этого, дату доставки, которая заносится после того, как покупка получена заказчиком. Заказ считается выполненным после того, как книга оплачена и отдана в руки покупателя — лично в магазине или отправлена средствами доставки. Неоплаченную книгу нельзя ни вынести из магазина, ни отправить по почте. В каждом заказе обязательно отмечается способ оплаты и статус заказа. Способы оплаты бывают: наличными, чеком и посредством кредитной карты. Статусов заказа четыре: 1) заказ подлежит отправке; 2) заказ будет получен лично; 3) заказ отправлен; 4) заказ получен. В заказе фигурирует только один покупатель, один продавец, дата заказа и доставки, способ оплаты и статус; однако заказ может формироваться из нескольких книг.

В настоящее время вся работа с заказами выполняется с помощью бумажных бланков, в том числе проверяется факт отправки товара (если он должен быть отправлен) и ведется учет проданного товара. Любая добавленная к заказу книга вычеркивается из инвентарного списка. Этот процесс весьма утомителен и не всегда достаточно эффективен. При этом нельзя быть уверенным, что никто ничего не напутает и не ошибется. Управляющий хочет, чтобы проданная книга осталась в списке книг, но с пометкой о том, что она уже продана.

В магазине продается примерно 20 книг в день. Он открыт пять дней в неделю в течение 10 часов ежедневно. Одновременно за двумя при-

лавками работают один-два продавца, которые принимают плату, выдают покупки и обрабатывают заказы. В магазине всегда находится как минимум один управляющий. Управляющий предполагает приблизительно 10-процентный годовой прирост объема продаж. А значит, примерно такие же темпы роста наличного количества книг (а значит, и числа авторов) и покупателей.

Для эффективного обслуживания покупателей каждому работнику необходим доступ к централизованному источнику информации об авторах, имеющихся в магазине книгах, покупателях и заказах. В настоящее время работники берут эти сведения из каталожных карточек и списков. Часто в них содержатся устаревшие сведения, что ведет к ошибкам. Кроме того, вместо заполнения бумажных бланков у каждого работника должна быть возможность оперативного создания, учета и модификации заказов. Однако право модифицировать сведения об авторах, книгах и покупателях следует предоставить только управляющим.

---

2. Разобраться с целями создания системы, исходя из сценария.

Какую из целей можно оформить в *измеримых* величинах?

Например, в качестве одного из вариантов целей можно предложить (в тексте сценария цели отмечены красным цветом):

- **... облегчить и сделать более эффективными управление складскими запасами, учет заказов и продаж.**
- **... требуется возможность доступа и, при необходимости, модификации сведений о каждом работнике.**
- **... магазин также собирает сведения о покупателях...**

• ... необходимо регистрировать продажи, отслеживая заказ с момента его приема продавцом до оплаты.

• ... каждому работнику необходим доступ к централизованному источнику информации об авторах, имеющихся в магазине книгах, покупателях и заказах.

Определение объема и типов данных

3. Определить категории данных, обнаруженные при знакомстве со сценарием.

4. Записать типы сведений, которые необходимо учитывать для каждой категории данных, выделенной в п. 3. Можно здесь сразу же сформулировать соображения по типам данных.

5. Записать текущий объем данных для каждой категории данных, выделенной в п. 3.

6. Для каждой выделенной в п. 3 категории данных записать ожидаемую тенденцию роста.

Так, в частности, могут выглядеть результаты анализа типов и объема данных (в тексте сценария отмечены зеленым цветом):

• ... общее число книг в магазине, как правило, не превышает нескольких тысяч.

• ... сведения для любой книги состоят из ее названия, автора, стоимости, рекомендованной розничной цены и оценки состояния.

• ... следует учитывать каждый экземпляр книги по отдельности...

• Управляющий записывает краткие сведения о каждом авторе...

• В настоящее время у управляющего имеются сведения примерно о 2500 авторах.

• Штат магазина состоит из 12 работников...

• В течение ближайших нескольких лет управляющий планирует нанимать дополнительно по одному работнику в год.

• В сведения о сотруднике входят имя, фамилия, адрес, номер телефона, дата рождения, дата приема на работу и название занимаемой в магазине должности.

• ... необходимо вести учет числа и вида книг, продаваемых каждым работником.

• ... из личных сведений о покупателе необходимыми считаются лишь имя и фамилия.

• На данный момент в список управляющего занесено 2000 покупателей...

• В магазине продается примерно 20 книг в день. Он открыт пять дней в неделю в течение 10 часов ежедневно.

• Управляющий предполагает приблизительно 10-процентный годовой прирост объема продаж. А значит, примерно такие же темпы роста наличного количества книг (а значит, и числа авторов) и покупателей.

Определение способа использования данных

7. Определить категории пользователей на основе сценария.

8. Записать число пользователей, относящихся к каждой категории, выделенной в п. 7. Нужно отметить настоящее число пользователей и число в ближайшем будущем.

9. Записать задачи каждой категории пользователей, выделенной в п. 7.

В тексте сценария категории, задачи и количество пользователей выделены желтым цветом:

• ... управляющий ведет всю документацию по продажам и инвентаризации...

• Управляющему хотелось бы иметь возможность добавить краткое, длиной всего в пару предложений, описание каждой оценки...

• ... управляющий назначает каждой книге уникальный идентификатор...

• Управляющий также записывает краткие сведения о каждом авторе...

• Управляющий также хотел бы добавлять краткую информацию об авторе (если она есть)...

• Штат магазина состоит из 12 работников... В течение ближайших нескольких лет управляющий планирует нанимать дополнительно по одному работнику в год.

• Как управляющему, так и его помощнику требуется возможность доступа и, при необходимости, модификации сведений о каждом работнике.

• ... у управляющего может возникнуть желание добавить к списку должностей новые должности или изменить существующие, а также кратко описать обязанности для каждой должности (по крайней мере, некоторых из них).

• Ни у кого, кроме управляющего и его помощника, не должно быть доступа к сведениям о персонале.

• Управляющему также хотелось бы вести учет числа и вида книг, продаваемых каждым работником.

• ... магазин также собирает сведения о покупателях...

• Управляющему необходимо регистрировать продажи...

• Управляющий хочет, чтобы проданная книга осталась в списке книг, но с пометкой о том, что она уже продана.

• ... работают один-два продавца, которые ... обрабатывают заказы.

• ... каждому работнику необходим доступ к централизованному источнику информации об авторах, имеющихся в магазине книгах, покупателях и заказах.

• ... у каждого работника должна быть возможность оперативного создания, учета и модификации заказов. Однако право модифицировать сведения об авторах, книгах и покупателях следует предоставить только управляющим.

Определение бизнес-правил

10. Записать бизнес-правила, определенные на основе представленной в сценарии информации.

В тексте сценария бизнес-правила выделены голубым цветом:

• Для каждой книги управляющий записывает название, автора, ... стоимость, рекомендованную розничную цену и оценку состояния книги. Последний параметр оценивается следующими категориями: превосходное, отличное, хорошее, неплохое, плохое, книга повреждена.

• Название издательства, дата выхода и номер редакции указываются не всегда.

• ... год издания не может быть меньше 1600 и больше 2099...

• Идентификатор книги состоит из восьми символов — цифр и букв.

• В магазине иногда представлены несколько книг одного автора, а одну книгу иногда пишут несколько авторов.

• Сведения об авторе состоят из его имени, фамилии, года рождения и (в отдельных случаях)

года смерти. Из этой информации необходимо, по крайней мере, имя автора.

• В сведения о работнике входят имя, фамилия, адрес, номер телефона, дата рождения, дата приема на работу и название занимаемой в магазине должности.

• Штатное расписание магазина предусматривает должности: управляющего, помощника управляющего, продавца на полный рабочий день и продавца на неполный рабочий день.

• Сотрудник может занимать только одну должность.

• ... сведения о покупателях: имя, фамилия, номер телефона, почтовый адрес, купленные книги и дата покупки. ... необходимыми считаются лишь имя и фамилия.

• В каждом заказе необходимо указывать сведения о проданной книге, ее покупателе, оформившем покупку продавце, числе проданных экземпляров и дате заказа. А кроме этого, дату доставки, которая заносится после того, как покупка получена заказчиком.

• В каждом заказе обязательно отмечается способ оплаты и статус заказа. Способы оплаты бывают: наличными, чеком и посредством кредитной карты. Статусов заказа четыре: 1) заказ подлежит отправке; 2) заказ будет получен лично; 3) заказ отправлен; 4) заказ получен.

• В заказе фигурирует только один покупатель, один продавец, ... однако заказ может формироваться из нескольких книг.

• Управляющий хочет, чтобы проданная книга осталась в списке книг, но с пометкой о том, что она уже продана.

---

§1.3. Упражнение «*Разработка логической модели данных*»

Добавление таблиц

1. Основываясь на сценарии и результатах выполнения упражнения *«Определение требований к БД. Сценарий БД (книжный магазин)»* (§ 1.2), нарисовать таблицу для каждой категории данных. В первом приближении должно получиться пять таблиц: *Books*, *Authors*, *Employees*, *Customers*, *Orders*.

2. Определить связанные (или связывающие) таблицы по бизнес-правилам (п. 10 § 1.2). По сути, это будут некие подкатегории информации, которых просматривается четыре: *состояние книги*, *должность работника*, *способ оплаты заказа*, *статус заказа* (в сценарии за каждой такой подкатегорией просматривается *список* будущих значений).

В результате получаются еще четыре таблицы: *BookCondition*, *Positions*, *FormOfPayment*, *OrderStatus*.

3. Поскольку в бизнес-правилах указывается, что в одном заказе может быть несколько книг, целесообразно добавить еще одну таблицу *BooksOrders* для учета заказанных книг и реальных заказов от покупателей.

В итоге в базу данных добавляется 10 таблиц.

Определение столбцов таблиц

4. Для каждой категории данных в сценарии определена информация, входящая в категорию – эта информация и определяет столбцы будущей таблицы. С учетом этого попробовать определить столбцы каждой из 10 таблиц, добавленных в БД.

Один из возможных вариантов столбцов (и последующих диаграмм связей между таблицами) содержится в файле *Таблицы БД для книжного магазина.doc*:

Таблица	***Столбцы***
Books	**TitleID, Title, AuthorID, Publisher, PubDate, Edition, Cost, SRP, ConditionID, Sold**

```
BookCondition  ConditionID, ConditionName,
               Description

Authors        AuthorID, FirstName, LastName,
              YearBorn, YearDied, Description

Employees      EmployeeID,          FirstName,
               LastName, Address1, Address2,
               City, State, Zip, Phone, DOB,
               HireDate, PositionID

Positions      PositionID, Title, JobDescrip

Customers      CustomerID,          FirstName,
               LastName,   Phone,   Address1,
               Address2, City, State, Zip

Orders         OrderID,           CustomerID,
               EmployeeID, Amount, OrderDate,
               DeliveryDate,PaymentID,
               StatusID

OrderStatus      StatusID, StatusDescrip

FormOfPayment  PaymentID, PaymentDescrip

BooksOrders    OrderID, TitleID
```

Замечания.

• Сокращения в таблицах означают: DOB (Date Of Birth) – дата рождения, SRP (Suggested Retail Price) – предлагаемая розничная цена.

• В таблицах довольно много идентификаторов (AuthorID, CustomerID, TitleID и т.п.) – это естественно для ссылок на связанные таблицы и вообще для ссылок на каждую строку отдельно.

• В таблицах нет столбца со сведениями о купленных книгах и датах покупок. Дело в том, что каждый покупатель может

купить *несколько* книг. Это приводит к мысли о необходимости отдельной таблицы. Однако при этом будет дублироваться информация, уже существующая в БД. Таким образом, лучше отдельную таблицу не создавать, а информацию о купленных книгах и датах покупок выводить с помощью представления или специализированного запроса.

Определение связей между сущностями

5. Определить связи, например, для таблицы Books (по сценарию БД для книжного магазина, § 1.2). Сначала определить прямые связи типа Books → BookCondition, Authors → Books, BooksOrders → Books и т.п.

Замечание. Прямой связи между таблицами Books и Orders нет! Дело в том, что между этими таблицами существует отношение «многие к многим», что приводит к необходимости создания «связующей» таблицы BooksOrders.

6. Соединить все связанные таблицы линиями (желательно, непересекающимися), в результате чего получится неконкретизированная схема данных (рис. 1).

7. Определить типы связей («один к одному», «один к многим», «многие к многим»); концы связей, соответствующие «одному», обозначить символом «1», концы, соответствующие «многим», – символом «∞» (рис. 2). Тип связи достаточно очевиден из сценария (один автор – много книг, один служащий – много заказов и т.п.).

8. Для связи «многие к многим» между таблицами Books и Authors создать соединяющую таблицу BookAuthors со столбцами AuthorID и TitleID.

Удалить связь между таблицами Books и Authors.

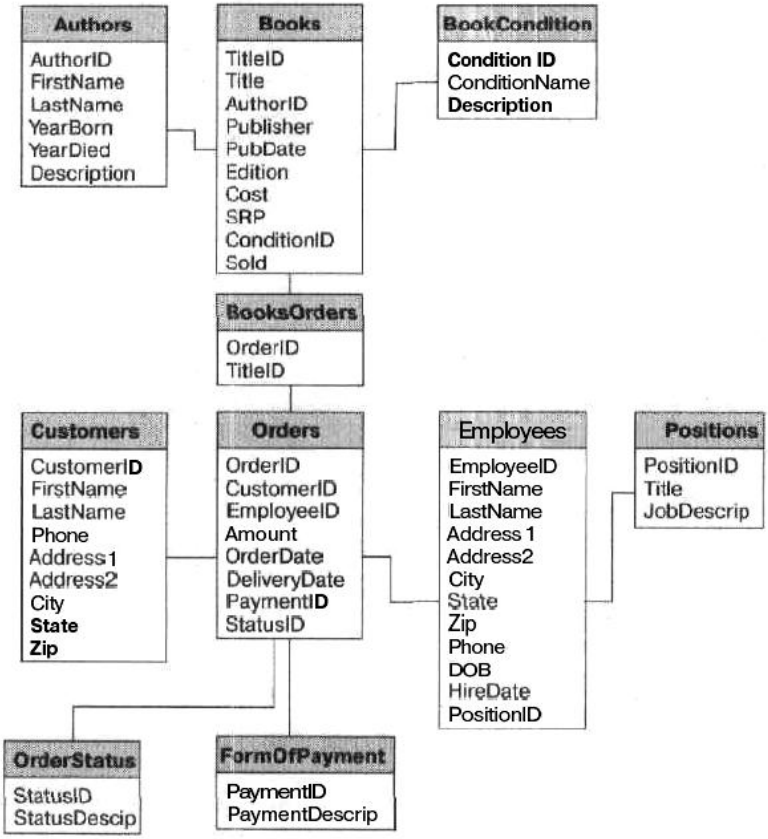


Рис. 1 — Структура БД для книжного магазина

Удалить столбец AuthorID из таблицы Books, поскольку связь «книга – автор» выражена через таблицу BookAuthors.

Установить связи (видимо, «один к многим») между таблицами Authors и BookAuthors, Books и BookAuthors (рис. 3).

Замечание. Рисунки 1, 2 и 3 находятся в том же файле, что и столбцы таблиц (*Таблицы БД для книжного магазина.doc*).

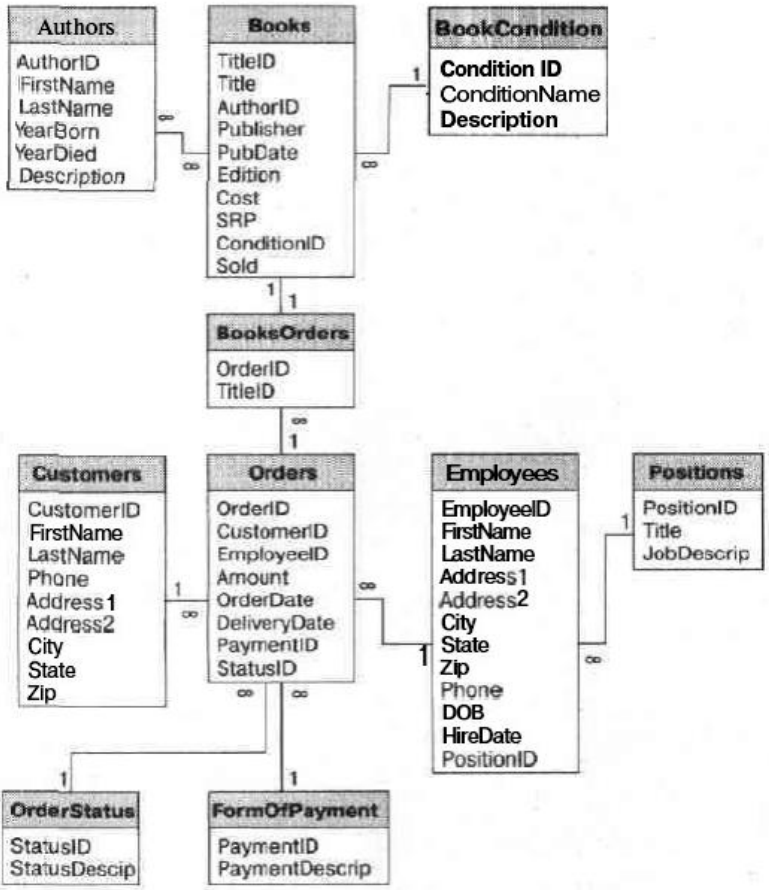


Рис. 2 — Структура БД с типами связей

Определение ограничений, налагаемых на данные

9. Вспомнить бизнес-правило типа

«Сведения о книге должны состоять из заглавия, автора, стоимости, предполагаемой розничной цены, оценки состояния и уникального идентификатора»

(п. 10, § 1.2).

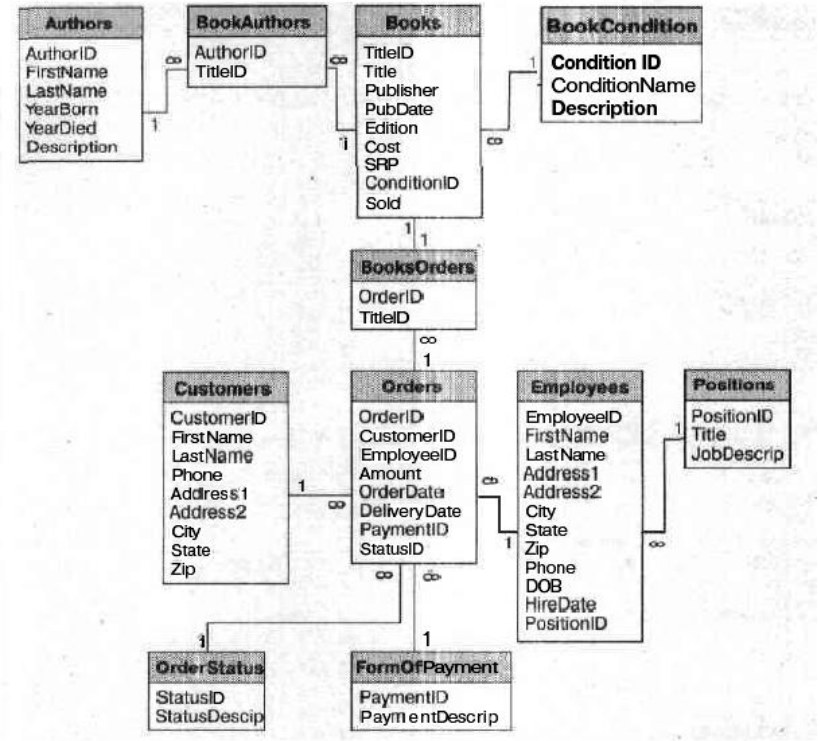


Рис. 3 — Таблица BookAuthors в структуре БД

Найти объект, к которому применимо это бизнес-правило, если такой объект существует. (Оказывается, существуют сразу *два* таких объекта – таблица Books и таблица BookAuthors).

Вывести ограничения, вытекающие из этого бизнес-правила.

Аналогичную работу провести для каждой таблицы (а в идеале – и для каждого столбца каждой таблицы).

§1.4. Упражнение
«Создание и управление базой данных»

Общее замечание. В данном упражнении рассматривается комплекс технологических приемов создания БД и управления ею (конкретно, два способа создания БД, просмотр БД, увеличение размера основного файла, добавление дополнительного файла данных, удаление дополнительного файла данных).

Эти приемы шире, чем те, которые необходимы для решения утилитарной задачи – создания базы данных книжного магазина (BookShopDB).

1. Запустить Management Studio и подключиться к локальному серверу.

2. Ввести следующий код T-SQL в окне редактора запросов (комментарии, делящие код на части, вводить не нужно):

```
USE master
GO
CREATE DATABASE BookShopDB
```

Эти команды создают БД BookShopDB (книжный магазин).

```
ON PRIMARY
(
NAME = Bookshop_dat,
FILENAME = 'C:\Program Files\Microsoft SQL
Server\MSSQL.1\MSSQL\Data\Bookshop.mdf',
SIZE = 4,
MAXSIZE = 10,
FILEGROWTH = 1
)
```

Эти команды определяют основной файл БД: логическое имя (для операторов T-SQL) Bookshop_dat, путь и имя файла C:\ Program Files\ Microsoft SQL Server\ MSSQL.1\ MSSQL\ Data\ Bookshop.mdf, первоначальный размер 4 Мб, максимальный размер 10 Мб, инкремент роста файла 1 Мб.

```
LOG ON
(
NAME = Bookshop_log,
```

```
FILENAME = 'C:\Program Files\Microsoft SQL
Server\MSSQL.1\MSSQL\Data\Bookshop.ldf',
SIZE = 2,
MAXSIZE = 5,
FILEGROWTH = 1
)
GO
```

Эти команды определяют файл журнала транзакций: логическое имя (для операторов T-SQL) Bookshop_log, путь и имя файла C:\ Program Files\ Microsoft SQL Server\ MSSQL.1\ MSSQL\ Data\ Bookshop.ldf, первоначальный размер 2 Мб, максимальный размер 5 Мб, инкремент роста файла 1 Мб.

Для сокращения ввода кода можно использовать текстовый файл *Создание БД.txt*.

3. Выполнить запрос по ! ***Выполнить запрос*** и убедиться, что база данных создана (в соответствующей папке появились два файла с нужными именами и нужного размера).

Просмотр базы данных BookShopDB

4. Найти базу данных BookShopDB в древовидной структуре обозревателя объектов Management Studio: ***Базы данных*** → ***BookShopDB.***

5. Просмотреть свойства БД по контекстному меню базы данных ➢***Свойства.*** В диалоговом окне □ ***Свойства базы данных*** найти имя БД, имена и размеры файлов, авторасширения размеров, ограничители размеров.

Удаление базы данных BookShopDB

6. В окне редактора запросов ввести оператор T-SQL

```
DROP DATABASE  BookShopDB
```

Выполнить запрос и убедиться, что оба файла базы данных удалены.

Создание базы данных с помощью Management Studio

7. Поместить курсор в узел ***Базы данных*** дерева объектов и в контекстном меню этого узла выполнить пункт ➢***Создать базу данных...***

В появившемся диалоговом окне ☐***Создание базы данных*** ввести только имя БД (BookShopDB) и нажать ❑***OK*** (все остальные параметры можно взять по умолчанию).

Убедиться, что в соответствующей папке созданы два файла: *BookShopDB.mdf* и *BookShopDB.ldf*.

Убедиться, что БД BookShopDB добавлена в список баз в дереве объектов(для этого, возможно, придется выполнить пункт ➢***Обновить*** контекстного меню узла ***Базы данных***).

Можно, как в п. 5, ознакомиться со свойствами созданной базы данных.

Увеличение размера базы данных

8. Вызвать окно свойств БД BookShopDB (см. п. 5) и во вкладке 🗀***Файлы*** увеличить начальный размер файла данных на 1 Мб.

Убедиться, что изменение размера файла прошло.

Добавление второго файла данных в базу

9. Во вкладке 🗀***Файлы*** диалогового окна ☐***Свойства базы данных*** нажать кнопку ❑***Добавить*** и в пустую строку ввести BookShopDB2 в качестве логического имени. Нажать ❑***OK*** .

(Можно одновременно и изменить размер этого файла данных).

Убедиться, что создан дополнительный файл *BookShopDB2.ndf*.

Удаление второго файла данных из базы

10. Выбрать во вкладке 🗀***Файлы*** диалогового окна ☐***Свойства базы данных*** строку с логическим именем BookShopDB2 и нажать кнопку ❑***Удалить***, а далее ❑***OK*** .

Убедиться, что соответствующий файл удален.

§1.5. Упражнение
«Определение типов данных для столбцов»

Просмотр существующих таблиц, столбцов и их типов данных

1. Открыть Management Studio (если он уже не открыт) и по дереву объектов добраться до таблиц базы данных Northwind (***Базы данных*** → ***Northwind*** → ***Таблицы***).

2. Раскрыть список столбцов таблицы Employees через ***Таблицы*** → ***dbo.Employees*** → ***Столбцы***. Изучить список типов данных столбцов и обратить внимание, что *сразу же* определяется размер столбца и условие на пустое значение (NULL).

3. Ознакомиться с типами данных для столбцов других таблиц БД Northwind.

Определение типов данных для таблицы Authors базы данных BookShopDB

4. Анализируя список столбцов таблицы Authors (см. п. 4 § 1.3), можно «прикинуть» типы данных для ее столбцов, например:

```
AuthorID    smallint        not NULL
FirstName   varchar(30)     NULL
LastName    varchar(30)     not NULL
YearBorn    char(4)         NULL
YearDied    char(4)         NULL
Description varchar(200)    NULL
```

Соображения по типам данных:

- Для AuthorID желательна не только уникальность, но и *генерируемость* сервером, в связи с чем целесообразно «навесить» на этот столбец свойство IDENTITY. Однако свойство IDENTITY работает только для типов столбца «целочисленный» (int, smallint, tinyint) и «десятичный» (decimal). В нашем случае нужно выбирать тип «целочисленный», так как десятичные дроби (тип «десятичный») для идентификации авторов не нужны. По сценарию нужно хранить сведения примерно о 2500 авторах, поэтому можно использовать тип smallint, который дает 32767 различных идентификаторов.

- Для FirstName и LastName нужны символьные типы «с запасом» – например, до 30 символов, поэтому можно планировать тип varchar(30) для обоих столбцов.
- Для YearBorn и YearDied можно предполагать по четыре символа года (поскольку нужны только *годы* рождения и смерти автора, но не даты). Однако в категории «дата и время» такого типа нет, поэтому стоит использовать *символьный* тип данных, например, varchar(4).
- Для Description лучше задать *ограничение* (например, 200 символов), поскольку в сценарии сказано «1-2 предложения»; таким образом, «вырисовывается» тип varchar(200).

5. Создать в БД BookShopDB таблицу Authors и создать в ней столбцы с указанными в п. 4 типами.

Создание таблицы можно выполнить через пункт ➢***Создать таблицу*** контекстного меню узла ***Таблицы*** дерева объектов.

При заполнении типов столбцов не следует забывать *сразу* корректировать свойства соответствующего столбца (во вкладке 🗀***Свойства столбцов*** внизу экрана)**.**

6. Сохранить таблицу нажатием кнопки  (Сохранить таблицу Table_1), а при сохранении переименовать ее в Authors.

Замечание. Сохранение таблицы может занять ощутимое время.

7. Если позволит время, спроектировать типы данных для столбцов оставшихся таблиц БД BookShopDB. Обратить внимание на совпадение или *близость* типов *будущих* первичных и внешних ключей.

§1.6. Упражнение
«Создание таблиц и управление ими»

Создание таблицы Authors в базе данных BookShopDB

1. В окне редактора запросов Management Studio ввести следующий код T-SQL:

```
USE BookShopDB
CREATE TABLE Authors
(
AuthorID SMALLINT IDENTITY(101,1) NOT NULL,
FirstName VARCHAR(30)  DEFAULT 'unknown',
LastName VARCHAR(30) NOT NULL,
YearBorn CHAR(4)  DEFAULT 'N/A',
YearDied CHAR(4)  DEFAULT 'N/A',
Description VARCHAR(200) DEFAULT 'N/A'
)
```

По сценарию хотя бы фамилия автора должна быть введена, в силу чего столбец LastName *требует* ввода значения. Конструкция NOT NULL используется для запрета ввода пустых значений (в частности, для запрета отсутствия ввода), но, где это возможно, применены значения по умолчанию (DEFAULT). Для столбца AuthorID определено свойство IDENTITY(101,1); таким образом, первой строке в таблице Authors будет присвоено AuthorID:=101, второй – AuthorID:=102 и т.д.

Вышеприведенный код можно не вводить (как и последующие), а скопировать из текстового файла *Создание таблиц.txt*.

2. Исполнить запрос кнопкой ! и убедиться, анализируя дерево объектов, что таблица Authors создана (возможно, придется предварительно выполнить пункт ➢***Обновить*** контекстного меню узла ***Таблицы*** дерева объектов).

Обратить внимание, что свойства таблицы совпадают с объявленными в операторе CREATE TABLE (свойства доступны через пункт ➢***Свойства*** контекстного меню узла ***dbo.Authors*** дерева объектов).

Создание таблиц BookAuthors и BookCondition в БД BookShopDB

3. Открыть в Блокноте файл *Создание таблиц.txt* и скопировать оттуда в окно редактора запросов следующий код T-SQL:

```
USE BookShopDB
CREATE TABLE BookAuthors
(
AuthorID SMALLINT  NOT NULL,
TitleID CHAR(8) NOT NULL
)

CREATE TABLE BookCondition
(
ConditionID TINYINT NOT NULL,
ConditionName CHAR(10) NOT NULL,
Description VARCHAR(50)  DEFAULT 'N/A'
)
```

В отличие от п. 1 свойство IDENTITY для таблицы BookCondition указывать не нужно: она маленькая, и можно вручную присвоить уникальное значение столбцу ConditionID. BookAuthors – *соединяющая* таблица (см. п. 8 § 1.3), поэтому ее схема определяется *другими* таблицами (Authors и Books).

4. Выполнить запрос и убедиться, что нужные таблицы с нужными столбцами созданы (возможно, потребуется обновление списка таблиц БД BookShopDB в окне дерева объектов).

5. Попробовать создать таблицу через диалоговый интерфейс Management Studio: в контекстном меню узла дерева объектов ***Таблицы*** выполнить пункт ➢***Создать таблицу.***

Убедиться, что далеко не все свойства таблицы могут так задаваться (см., например, UNIQUE, IDENTITY).

Создание остальных таблиц в БД BookShopDB

6. Можно скопировать *весь* файл *Создание таблиц.txt* в окно редактора запросов и попробовать создавать таблицы *по частям*, выделяя части запроса и нажимая кнопку ! (Выполнить запрос).

Код, который должен быть в результате выполнен, следующий:

```
    CREATE TABLE Books
    (
    TitleID CHAR(8) UNIQUE NOT NULL,
    Title VARCHAR(130)  DEFAULT 'unknown',
    AuthorID SMALLINT  NOT NULL,
    Publisher VARCHAR(50)  DEFAULT 'N/A',
    PubDate DATETIME  DEFAULT '01/01/1900',
    Edition TINYINT  DEFAULT 0,
    Cost SMALLMONEY DEFAULT 0,
    SRP SMALLMONEY DEFAULT 0,
    ConditionID TINYINT NOT NULL,
    Sold BIT NOT NULL
    )

    CREATE TABLE Employees
    (
EmployeeID SMALLINT IDENTITY(2101,1) NOT NULL,
FirstName VARCHAR(30) NOT NULL ,
LastName VARCHAR(30) NOT NULL,
Address1 VARCHAR(30) NOT NULL ,
Address2 VARCHAR(30) ,
City VARCHAR(20) NOT NULL ,
State CHAR(2) NOT NULL,
Zip CHAR(6) NOT NULL ,
Phone CHAR(15) NOT NULL ,
DOB  DATETIME  NOT NULL,
HireDate DATETIME  NOT NULL,
PositionID TINYINT NOT NULL
    )

    CREATE TABLE Customers
    (
CustomerID SMALLINT IDENTITY(4101,1) NOT NULL,
FirstName VARCHAR(30)  DEFAULT 'unknown',
LastName VARCHAR(30)  DEFAULT 'unknown',
Address1 VARCHAR(30) DEFAULT 'N/A' ,
```

```
Address2 VARCHAR(30) ,
City VARCHAR(20) ,
State CHAR(2) ,
Zip CHAR(6) ,
Phone CHAR(15) NOT NULL
    )

    CREATE TABLE Positions
    (
    PositionID TINYINT NOT NULL,
    Title CHAR(20) NOT NULL,
    JobDescrip VARCHAR(350)  DEFAULT 'N/A'
    )

    CREATE TABLE FormOfPayment
    (
    PaymentID TINYINT NOT NULL,
    PaymentDescrip VARCHAR(250) DEFAULT 'N/A'
    )

    CREATE TABLE Orders
    (
OrderID SMALLINT  IDENTITY(12101,1) NOT NULL,
CustomerID SMALLINT  NOT NULL,
EmployeeID SMALLINT  NOT NULL,
Amount TINYINT NOT NULL ,
OrderDate DATETIME  NOT NULL,
DeliveryDate DATETIME  NOT NULL,
PaymentID TINYINT NOT NULL,
StatusID TINYINT NOT NULL
    )

    CREATE TABLE OrderStatus
    (
    StatusID TINYINT NOT NULL,
    StatusDescrip VARCHAR(150) DEFAULT 'N/A'
    )
```

```
CREATE TABLE BooksOrders
(
OrderID SMALLINT  NOT NULL,
TitleID CHAR(8) NOT NULL
)
```

Попробовать разобраться в структуре всех таблиц (см. также рис. 1) и, возможно, исправить какие-либо ошибки или неточности.

В итоге должно быть создано 11 таблиц.

Сохранять текст запроса не нужно – он есть в виде файла *Создание таблиц.txt*, а таблицы сохранятся автоматически.

§1.7. Упражнение *«Определение свойств, гарантирующих целостность данных»*

Определение свойств таблицы Employee из базы данных Pubs

1. В редакторе запросов Management Studio ввести операторы T-SQL:

```
USE Pubs
GO
sp_help Employee
```

Исполнить операторы нажатием !.

В итоге во вкладке 🗀***Результаты*** выведутся сведения о таблице Employee базы данных Pubs.

Найти в этих сведениях информацию о типах данных столбцов, возможности ввода пустых значений, определениях DEFAULT, свойстве IDENTITY, индексах и ограничениях.

Определение свойств таблицы Publishers

2. В редакторе запросов ввести, а затем выполнить команду T-SQL:

```
sp_help Publishers
```

По вкладке 🗀***Результаты*** проанализировать сведения о типах данных, возможности ввода пустых значений, определениях DEFAULT, свойстве IDENTITY, индексах и ограничениях.

Постараться, например, ответить на следующие вопросы:

- Какого типа ограничения наложены на столбцы?
- Какой столбец содержит идентификатор?
- Для каких столбцов определен символьный тип данных?
- Какие столбцы допускают пустые значения?

Определение свойств таблицы Titles

3. В редакторе запросов ввести, а затем выполнить команду T-SQL:

```
sp_help Titles
```

На вкладке 🗀***Результаты*** выведутся сведения таблице Titles, в которых найти информацию о типах данных, возможности ввода пустых значений, определениях DEFAULT, свойстве IDENTITY, индексах и ограничениях.

Постараться, например, ответить на вопрос:

- Есть ли пользовательский тип данных в таблице Titles?

§1.8. Упражнение
«Добавление ограничений в существующие таблицы»

Добавление ограничения PRIMARY KEY к таблице Authors базы данных BookShopDB

1. По бизнес-правилам сценария (п. 10 § 1.2) и структуре БД найти столбец (или столбцы) таблицы Authors, для которого следует определить ограничение PRIMARY KEY. Легко установить, что это должен быть столбец AuthorID.
2. В окне редактора запросов ввести код T-SQL:

```
USE BookShopDB
ALTER TABLE Authors
ADD CONSTRAINT authors_pk
                PRIMARY KEY (AuthorID)
```

Исполнить этот запрос и добиться сообщения об успешном завершении.

Текст этого и последующих запросов можно взять из текстового файла *Добавление ограничений PRIMARY KEY.txt*.

Добавление ограничения PRIMARY KEY к таблице BookAuthors

3. Обратиться к бизнес-правилам и структуре БД, чтобы найти столбец (или столбцы) таблицы BookAuthors, для которого следует определить ограничение PRIMARY KEY. По всей видимости, нужно определять это ограничение для *комбинации* столбцов AuthorID и TitleID.

4. Ввести в окно редактора запросов следующий код T-SQL:

```
ALTER TABLE BookAuthors
ADD CONSTRAINT bookauthors_pk
            PRIMARY KEY (AuthorID, TitleID)
```

Исполнить запрос и добиться сообщения об успешном завершении.

Добавление ограничения PRIMARY KEY к остальным таблицам БД BookShopDB

5. Открыть в Блокноте файл *Добавление ограничений PRIMARY KEY.txt* и скопировать необходимые операторы T-SQL (или все операторы).

Код, который должен быть в результате скопирован, следующий:

```
ALTER TABLE Books
ADD CONSTRAINT books_pk PRIMARY KEY (TitleID)

ALTER TABLE BookCondition
ADD CONSTRAINT bookcond_pk
            PRIMARY KEY (ConditionID)

ALTER TABLE Customers
ADD CONSTRAINT customers_pk
            PRIMARY KEY (CustomerID)

ALTER TABLE Orders
ADD CONSTRAINT orders_pk
            PRIMARY KEY (OrderID)
```

```
ALTER TABLE Employees
ADD CONSTRAINT employees_pk
                PRIMARY KEY (EmployeeID)

ALTER TABLE Positions
ADD CONSTRAINT positions_pk
                PRIMARY KEY (PositionID)

ALTER TABLE OrderStatus
ADD CONSTRAINT orderstatus_pk
          PRIMARY KEY (StatusID)

ALTER TABLE FormOfPayment
ADD CONSTRAINT payment_pk
                PRIMARY KEY (PaymentID)

ALTER TABLE BooksOrders
ADD CONSTRAINT booksorders_pk
                PRIMARY KEY (OrderID, TitleID)
```

Разобраться в формировании ограничений PRIMARY KEY для остальных таблиц (в результате такого формирования в каждой таблице должно быть по одному столбцу с первичным ключом, а в таблицах BookAuthors и BooksOrders первичный ключ создается для *двух* столбцов).

Выполнить получившийся запрос по .

Добавление ограничения FOREIGN KEY к таблице BookAuthors

6. Обратиться к бизнес-правилам (п. 10 § 1.2) и структуре БД, чтобы найти столбец (или столбцы) таблицы BookAuthors, для которого следует определить ограничение FOREIGN KEY. По всей видимости, нужно определять это ограничение и для столбца AuthorID, и для столбца TitleID, так как таблица является связывающей.

7. В связи с этим ввести в окно редактора запросов и исполнить следующий код T-SQL:

```
USE BookShopDB

ALTER TABLE BookAuthors
ADD CONSTRAINT authorid_fk
            FOREIGN KEY (AuthorID)
            REFERENCES Authors (AuthorID)

ALTER TABLE BookAuthors
ADD CONSTRAINT titleid_fk
            FOREIGN KEY (TitleID)
            REFERENCES Books (TitleID)
```

Можно не вводить код, а воспользоваться заготовкой из текстового файла *Добавление ограничений FOREIGN KEY.txt.*

Добавление ограничения FOREIGN KEY к остальным таблицам

8. По структуре БД можно убедиться, что это таблицы Books (одно ограничение), BooksOrders (два ограничения), Orders (четыре ограничения) и Employees (одно ограничение).

Для ввода соответствующего кода T-SQL удобно пользоваться заготовкой (п. 7) и, возможно, добавлять ограничения по частям (выделяя и исполняя части кода):

```
ALTER TABLE Books
ADD CONSTRAINT conditionid_fk
    FOREIGN KEY (ConditionID)
    REFERENCES BookCondition (ConditionID)

ALTER TABLE Orders
ADD CONSTRAINT customerid_fk
         FOREIGN KEY (CustomerID)
         REFERENCES Customers (CustomerID)

ALTER TABLE Orders
ADD CONSTRAINT employeeid_fk
         FOREIGN KEY (EmployeeID)
         REFERENCES Employees (EmployeeID)

ALTER TABLE Orders
```

```
ADD CONSTRAINT paymentid_fk
      FOREIGN KEY (PaymentID)
      REFERENCES FormOfPayment (PaymentID)

ALTER TABLE Orders
ADD CONSTRAINT statusid_fk
         FOREIGN KEY (StatusID)
         REFERENCES OrderStatus (StatusID)

ALTER TABLE Employees
ADD CONSTRAINT positionid_fk
         FOREIGN KEY (PositionID)
         REFERENCES Positions (PositionID)

ALTER TABLE BooksOrders
ADD CONSTRAINT ordertitleid_fk
               FOREIGN KEY (TitleID)
               REFERENCES Books (TitleID)

ALTER TABLE BooksOrders
ADD CONSTRAINT orderid_fk
               FOREIGN KEY (OrderID)
               REFERENCES Orders (OrderID)
```

Замечание. Результат можно проверить по списку объектов – узлы ***Столбцы*** и ***Ключи***. А еще лучше – создать диаграмму БД по контекстному меню ➢***Создать диаграмму*** узла дерева объектов ***BookShopDB → Диаграммы базы данных*** и сравнить ее с приведенной на рис. 3.

Добавление ограничения CHECK к таблицам Authors, Books и Customers

9. Анализируя бизнес-правила и структуру БД, найти в таблицах Authors и Books столбцы, для которых надо вводить ограничения CHECK. Видимо, это должны быть столбцы, связанные с годами рождения и смерти автора (каждый символ значения столбца должен быть цифрой, причем первый символ должен быть «1» или «2») и с датой выпуска книги (с 1 января 1600 г. до 31 декабря 2099 г.).

Указанные ограничения должны обеспечить доменную целостность БД.

10. Ту же работу провести для таблицы Customers (там нужно, чтобы имя и фамилия покупателя одновременно не были незаданными, то есть, чтобы не было значения ‘unknown’ одновременно для FirstName и LastName).

11. Для реализации п. 10 ввести в окно редактора запросов и исполнить следующий код T-SQL:

```
USE BookShopDB
ALTER TABLE Customers
ADD CONSTRAINT checknames_ck
CHECK (FirstName NOT LIKE 'unknown' OR
       LastName NOT LIKE 'unknown')
```

Убедиться, что ограничение CHECK создано – по открытию группы ***Ограничения*** дерева объектов (***Базы данных*** → ***BookShopDB*** → ***Таблицы*** → ***Customers*** → ***Ограничения***) или через выполнение системной хранимой процедуры

```
sys.sp_helpconstraint Customers
```

12. Выполнить ту же работу для таблиц Authors и Books (конечно, для скорости можно использовать заготовку запроса из файла *Добавление ограничений CHECK.txt*):

```
ALTER TABLE Authors
ADD CONSTRAINT authors1_ck
CHECK (YearBorn LIKE '[1-2][0,6-9][0-9][0-9]'
       OR (YearBorn = 'N/A'))

ALTER TABLE Authors
ADD CONSTRAINT authors2_ck
CHECK (YearBorn NOT LIKE '[1][0][0-9][0-9]')

ALTER TABLE Authors
ADD CONSTRAINT authors3_ck
CHECK (YearBorn NOT LIKE '[2][6-9][0-9][0-9]')

ALTER TABLE Authors
ADD CONSTRAINT authors4_ck
CHECK (YearDied LIKE '[1-2][0,6-9][0-9] [0-9]'
       OR (YearDied = 'N/A'))
```

```
ALTER TABLE Authors
ADD CONSTRAINT authors5_ck
CHECK (YearDied NOT LIKE '[1][0][0-9][0-9]')

ALTER TABLE Authors
ADD CONSTRAINT authors6_ck
CHECK (YearDied NOT LIKE '[2][6-9][0-9][0-9]')

ALTER TABLE Books
ADD CONSTRAINT books_ck
CHECK ((PubDate >= 01/01/1600) AND
       (PubDate <= 31/12/2099)OR
       (PubDate = '01/01/1900'))
```

Выполнить запрос и убедиться, что все предполагаемые ограничения созданы.

§ 2. ПРАКТИЧЕСКОЕ ЗАНЯТИЕ 2
Язык Transact SQL

Занятие предполагает знакомство с интерфейсом и основными возможностями редактора запросов (упражнение *«Работа с редактором запросов»* (§ 2.1)), а также работу с основными операторами T-SQL (упражнение *«Создание и исполнение операторов DDL, DCL и DML»* (§ 2.2)). Занятие завершается созданием простого сценария с переменными, функциями, выражениями, операторами цикла, блоками и т.п. (упражнение *«Создание сценариев с помощью синтаксических элементов языка T-SQL»* (§ 2.3)). Таким образом, осуществляется не только знакомство с элементами языка Transact SQL, но и складывается начальное представление о стиле программирования на этом языке.

§2.1. Упражнение
«Работа с редактором запросов»

1. После соединения с сервером и запуска Management Studio запустить редактор запросов по ❑***Создать запрос.***

Проверить, что результаты запроса будут выводиться в виде сетки:

➢***Запрос*** ➢***Отправить результаты в*** ➢***В виде сетки***

Подключить план выполнения запроса по

➢***Запрос*** ➢***Включить действительный план выполнения***

Подключить статистику выполнения запроса по

➢***Запрос*** ➢***Включить статистику клиента***

Поиск контекста (элемента оператора T-SQL) в файлах

2. Отобразить окно поиска по

➢***Правка*** ➢***Найти и заменить*** ➢***Найти в файлах…***

В диалоговом окне ☐***Поиск и замена*** установить:

Образец := FROM

Поиск в := C:\Program Files\MS SQL Server\90

{можно с помощью кнопки [...] «дойти» до папки нужного уровня или переключиться на другой диск}

☑ ***Включить вложенные папки***
Искать следующие типы файлов ⇩ *.sql
Выводить результаты в ⊙ Окно «Результаты поиска 1»
❑***Найти все***

Убедиться, что при достаточно быстром поиске можно легко переключаться на конкретные *запросы* (то есть файлы с расширением **.sql*), отображая их в окне редактора запросов (курсор при этом устанавливается на конструкцию FROM).

Замечание. Возможны повторные соединения с SQL Server, если результаты поиска находятся в других базах данных.

Отладка хранимой процедуры

3. В списке объектов окна обозревателя объектов найти базу данных ***Northwind***, а далее – узел ***Программирование → Хранимые процедуры***. Из *пользовательских* процедур выбрать ***dbo.CustOrderHist*** и попробовать выполнить следующие операции с контекстным меню:

➢***Изменить*** – для просмотра и изменения текста процедуры;

➢***Просмотреть зависимости*** – для отображения в отдельном окне дерева зависимостей («респондентов» и «корреспондентов») процедуры;

➢***Выполнить хранимую процедуру*** – для отображения в специальном окне (типа окна отладчика) параметров и результата выполнения процедуры.

По идее, если эта процедура есть среди пользовательских хранимых процедур, то при обращении к ней должна отмечаться ошибка (необходимо задать для запуска процедуры значение идентификатора покупателя @CustomerID). Если в окне выполнения процедуры указать значение @CustomerID (например, ***Значение*** := ANATR), то ошибка не появится, а во вкладке 🗀***Результаты*** будут выведены результаты выполнения процедуры (названия товаров, закупленных данным покупателем, и их общие количества).

Исполнение оператора SELECT

4. Ввести следующий оператор T-SQL в окно редактора запросов:

```
USE Northwind
SELECT * FROM Customers
```

Обратить внимание, что слова USE, SELECT, FROM отображаются синим цветом; так выделяются ключевые слова T-SQL.

5. Выполнить запрос по ***Выполнить запрос***; в результате должны открыться сразу четыре вкладки: ***Результаты***, ***Сообщения***, ***План выполнения***, ***Статистика клиента***.

Просмотреть результаты выполнения запроса; обратить внимание на комментарии в плане («всплывающие» и доступные по щелчку) – об использовании кластеризованного индекса, о сортировке и т.п. (в этой связи интересно взглянуть на план с точки зрения «стоимости» выполняемых операторов).

6. Выполнить запрос (п. 4) *еще два-три раза* и убедиться, что статистика *накапливается* и отмечаются тенденции, например, по времени выполнения (они будут изображаться в статистике стрелками ➔, ↑ и ↓).

Изменение оператора T-SQL

7. В редакторе запросов изменить слово **Customers** на **Custom**. При выполнении запроса изменится содержимое вкладки ***Сообщения***: появится сообщение об ошибке вида «Недопустимое имя объекта “Custom”».

Возвратить текст запроса в исходное состояние.

Важное замечание. При двойном щелчке мыши на тексте сообщения об ошибке выделится ошибочный оператор или его контекст.

8. Поработать с дополнительными возможностями выполнения запроса: включить вывод результата в виде текста (через ***Запрос Отправить результаты в В виде текста***); посмотреть предполагаемый план выполнения запроса (через ***Запрос Показать предполагаемый план выполнения***).

Факультативно. Попробовать запустить утилиту командной строки osql через ***Пуск Выполнить Открыть*** := cmd ***OK***

После появления приглашения **>** на ввод команды выполнить доверенное соединение с сервером с передачей имени базы данных и текста запроса:

```
osql -E <имя сервера> -d <имя БД>-Q "<запрос>"
```

В частности, запрос п. 4 для сервера LOCALSERV может быть оформлен следующей командной строкой:

```
osql -E LOCALHOST -d Northwind
     -Q "SELECT * FROM Customers"
```

При нажатии [Enter] отобразится результат выполнения запроса и приглашение **>**, на которое можно вводить новый запрос и т.д.

Описание синтаксиса утилиты всегда можно запросить по команде **`osql -?`**

§2.2. Упражнение «*Создание и исполнение операторов DDL, DCL и DML*»

Создание таблицы в базе данных Northwind

1. В окно редактора запросов ввести операторы T-SQL:

```
USE Northwind
CREATE TABLE Investors
(
InvestorID int NOT NULL,
FirstName varchar(30) NOT NULL,
LastName varchar(30) NOT NULL
)
```

Для ускорения ввода можно здесь и далее пользоваться текстовыми заготовками из файла *Операторы DDL, DCL и DML.txt*.

Обратить внимание, что ключевые слова **`USE`**, **`CREATE TABLE`**, **`int`**, **`varchar`** подсвечены синим цветом (можно по цвету определить безошибочность ввода).

Выполнить запрос по  и убедиться, что таблица Investors создана.

Открыть ее (например, по контекстному меню ***➢Открыть таблицу***) и убедиться, что она пустая.

Атрибуты столбцов легко проверить, раскрыв узел ***Столбцы***.

Модификация таблицы

2. В редакторе запросов установить курсор ниже предыдущего запроса (чтобы можно было что-то копировать из предыдущего) и ввести оператор изменения таблицы:

```
ALTER TABLE Investors
ADD InvestmentCode int NULL
```

Замечание. Команду **USE Northwind** можно не использовать, так как Northwind – активная база данных (список всех баз данных находится в панели инструментов, так что можно переключать БД по этому списку).

Выделить только эти две строки и нажать !.

Убедиться, что столбец InvestmentCode добавился в таблицу Investors.

Вывод сведений о таблице

3. Ниже введенного в п. 2 кода ввести вызов хранимой процедуры:

```
EXEC sp_help Investors
```

При исполнении запроса (выделить его) в вкладке 🗀***Результаты*** выводится в виде сетки информация о таблице Investors – похожая на ту, которая выводится по ➢***Открыть таблицу.***

В частности, указывается имя владельца таблицы (dbo – data base owner).

Предоставление и отзыв права доступа к таблице

4. В редакторе запросов ниже ввести оператор

```
GRANT SELECT
ON Investors
TO PUBLIC
```

Выделить оператор и выполнить его нажатием !.

Убедиться (например, с помощью пункта ➢***Свойства*** контекстного меню узла ***Investors*** обозревателя объектов; далее – вкладка 🗀***Разрешения*** диалогового окна свойств), что роли Public предоставлено право доступа к таблице Investors.

Это же можно проверить выполнением хранимой процедуры sp_helprotect:

```
EXEC sp_helprotect Investors
```

5. Выполнить оператор отзыва права доступа

```
REVOKE SELECT
ON Investors
TO PUBLIC
```

Убедиться (повторным вызовом диалогового окна свойств таблицы), что права доступа к таблице Investors у роли Public уже нет.

Извлечение данных

6. В окне редактора запросов ниже последней строки ввести операторы INSERT:

```
INSERT Investors VALUES (01, 'Amis','Baldwin', 103)
INSERT Investors VALUES (02, 'Jo' , 'Brown', 102)
INSERT Investors VALUES (03, 'Scott', 'Culp', 103)
INSERT Investors VALUES (04, 'Jon ', 'Grande', 103)
INSERT Investors VALUES (05, 'Lani', 'Ota', 102)
```

Выделить их и выполнить по **!**.

Убедиться, что во вкладке 🗀***Сообщения*** будет набор из пяти сообщений (то есть исполнение каждого оператора повлияло на одну строку).

7. Ввести и исполнить оператор SELECT:

```
SELECT * FROM Investors
```

Во вкладке 🗀***Результаты*** должны отобразиться пять строк, введенных в п. 6.

8. Набрать и выполнить следующий оператор SELECT:

```
SELECT FirstName, LastName FROM Investors
WHERE (InvestorID = 03 OR InvestorID = 05)
ORDER BY FirstName
```

Во вкладке 🗀***Результаты*** выводятся строки Lani Ota и Scott Culp (только имена и фамилии, отсортированные по имени в алфавитном порядке).

Модификация данных

9. Ввести и выполнить следующий оператор UPDATE:

```
UPDATE Investors
```

```
SET InvestmentCode = 101
WHERE InvestorID = 04
```

Во вкладке 🗀***Сообщения*** появится сообщение, что исполнение оператора повлияло на одну строку.

Проверка внесенных изменений – через контекстное меню ➢***Открыть таблицу*** либо через повторное выполнение оператора SELECT (см. п. 7):

```
SELECT * FROM Investors
```

10. Набрать и исполнить оператор DELETE:

```
DELETE FROM Investors
WHERE InvestorID = 04
```

Во вкладке 🗀***Сообщения*** появится сообщение, что исполнение оператора повлияло на одну строку.

Проверка удаления – через контекстное меню ➢***Открыть таблицу*** либо через повторное выполнение оператора SELECT (см. п. 7).

11. В заключение таблицу Investors можно удалить по

```
DROP TABLE Investors
```

Убедиться, что таблицы Investors нет по узлу ***Таблицы*** базы данных ***Northwind***.

Можно закрыть окно редактора запросов (в контекстном меню соответствующей *вкладки* выбрать ➢***Закрыть***), а в появившемся окне сохранения указать имя файла с расширением .sql. В результате все введенные в упражнении операторы (иными словами, сценарий) сохранятся.

§2.3. Упражнение *«Создание сценариев с помощью синтаксических элементов языка T-SQL»*

Создание таблицы в базе данных Northwind

В окне редактора запросов ввести код на языке T-SQL:

```
-- Выбрать базу данных
USE Northwind
GO
-- Создать таблицу
CREATE TABLE [New Table]
(ColumnA INT, ColumnB CHAR(3))
```

```
GO
SET NOCOUNT ON
GO
```

В этом фрагменте указывается используемая база данных и создается таблица с идентификатором New Table (из-за пробела и ключевого слова Table используются квадратные скобки) и идентификаторами столбцов ColumnA и ColumnB (они обычные, поэтому квадратные скобки указывать не нужно).

Замечания

- Оператор **SET NOCOUNT ON** блокирует вывод сообщений с числом строк, на которое повлияло исполнение текущего оператора T-SQL. Для «чистоты» в конце сценария следует указать **SET NOCOUNT OFF**, чтобы разрешить вывод этих сообщений для других запросов.
- Ключевое слово **GO** – не оператор T-SQL, а некая «команда», которую распознает анализатор запросов и утилита osql – как сигнал о том, что нужно отослать SQL Server текущий пакет операторов T-SQL.

Объявление и инициализация переменной и использование операторов управления ходом выполнения

2. После ввода операторов п. 1 ввести следующий код:

```
-- Объявить необходимую переменную
DECLARE @MyCounter INT
-- Инициализировать переменную
SET @MyCounter = 0
```

Таким образом, объявлена переменная MyCounter, определен ее тип как целый и присвоено значение 0.

3. Продолжить ввод кодов сценария:

```
/* Определить с помощью переменной число циклов*/
     WHILE (@MyCounter < 26)
     BEGIN
      -- Вставить в таблицу строку.
      INSERT INTO [New Table] VALUES
      -- С помощью переменной получить
      -- целочисленное значение для
      -- столбца ColumnA и сгенерировать уникальную
      -- букву для каждой строки.
```

```
    --Получить целочисленное значение символа "a"
    -- с помощью функции ASCII.
    -- Прибавить @MyCounter.
    -- С помощью функции CHAR
    -- преобразовать сумму обратно
    -- в символы @MyCounter,
    -- следующие после символа "a".
    (
    @MyCounter + 1,
    CHAR( ( @MyCounter + ASCII('a') ) )
    )
    /*Увеличить значение переменной для учета
       этой итерации цикла*/
    SET @MyCounter = @MyCounter + 1
  END
  GO
  SET NOCOUNT OFF
  GO
```

Замечание. В сценарий можно также включить оператор **TRUNCATE TABLE [New Table]** для сброса таблицы в начальное состояние.

Итак, сценарий завершен. Обратить внимание на применение операторов управления ходом выполнения – **WHILE** и **BEGIN-END**.

Значения строк в таблице определяются через выражения **@MyCounter+1** и **CHAR((@MyCounter+ASCII('a')))**, где **CHAR** и **ASCII** – функции.

Конечно, все из этого сценария вводить не нужно, а можно воспользоваться заготовкой из файла *Создание сценария на T-SQL.txt*.

Выполнение сценария с последующим удалением таблицы

4. Выполнить *полный* вариант сценария по **!**; в результате должна появиться вкладка 🗀***Сообщения*** с сообщением об успешном завершении сценария. (Если бы не использовался оператор **SET NOCOUNT ON**, то было бы выдано еще 26 сообщений с результатами подсчета строк).

5. Исполнить *одиночный* оператор SELECT:

```
SELECT * FROM [New Table]
```

По вкладке 🗀***Результаты*** можно убедиться, что в таблице 26 строк с увеличивающимся счетчиком и кодом литеры в столбцах.

6. Исполнить *одиночный* оператор

```
DROP TABLE "New Table"
```

(специально вместо квадратных скобок можно использовать двойные кавычки).

По вкладке 🗀***Сообщения*** убедиться, что команда выполнена успешно.

7. Закрыть вкладку со сценарием (в контекстном меню вкладки выбрать ➢***Закрыть***); если хочется, то можно сценарий сохранить в файле с расширением .sql.

§ 3. ПРАКТИЧЕСКОЕ ЗАНЯТИЕ 3
Выборка и модификация данных

Занятие посвящено, прежде всего, изучению возможностей оператора SELECT – основного и достаточно сложного оператора языка T-SQL. В упражнении *«Использование оператора SELECT для выборки данных»* (§ 3.1) выполняется выборка данных из таблицы по оператору SELECT в различных режимах. В упражнении *«Извлечение данных с помощью усложненных методик работы с запросами»* (§ 3.2) рассматриваются усложненные формы оператора SELECT (соединения, ключевое слово IN, подзапросы с ключевым словом ALL или ANY, подзапросы с ключевым словом EXISTS, сведение данных с помощью конструкций CUBE и ROLLUP).

Занятие завершается выполнением ряда действий по модификации данных в таблице (операторы INSERT, UPDATE, DELETE, TRUNCATE) – упражнение *«Модификация данных в БД SQL Server»* (§ 3.3).

§3.1. Упражнение *«Использование оператора SELECT для выборки данных»*

Общее замечание. Текстовые заготовки для операторов T-SQL, использующихся в упражнении, находятся в файле *Оператор SELECT.txt*.

Извлечение всех данных из таблицы Titles

1. В окне редактора запросов ввести операторы T-SQL:

```
USE pubs
SELECT * FROM titles
```

Звездочка (*) указывает, что надо извлечь *все* данные из таблицы Titles базы данных Pubs.

Замечание. Как указано во введении, предполагается, что используемая версия SQL Server не является чувствительной к регистру. Таким образом, использование идентификаторов

pubs, titles наряду с идентификаторами **Pubs, Titles** является корректным.

2. Выполнить запрос по ! и посмотреть результат во вкладке 🗀***Результаты.***

Получение данных из определенных столбцов таблицы Titles

3. Ввести следующий код T-SQL:

```
USE pubs
SELECT title_id, title, price, ytd_sales
FROM titles
```

Здесь извлекаются данные из четырех столбцов таблицы Titles (база данных Pubs): Title_id, Title, Price, Ytd_sales.

4. Выполнить запрос и посмотреть результат во вкладке 🗀***Результаты.***

Задание условия для результирующего набора

5. Ввести код запроса, который отбирает строки, значение поля Price которых больше 10 долл.:

```
USE pubs
SELECT title_id, title, price, ytd_sales
FROM titles
WHERE price > 10
```

6. Исполнить запрос и убедиться в правильности отбора.

Задание порядка вывода (сортировка) результирующего набора

7. Ввести код запроса, упорядочивающий результирующий набор, полученный в п. 5, сначала по цене (по убыванию), затем по заглавию (по возрастанию):

```
USE pubs
SELECT title_id, title, price, ytd_sales
FROM titles
WHERE price > 10
ORDER BY price DESC, title
```

8. Исполнить запрос и убедиться в правильности упорядочивания.

Группировка данных в результирующем наборе

9. Ввести код T-SQL, расширяющий предыдущий запрос:

```
USE pubs
SELECT type, AVG(price) AS AvgPrice
FROM titles
WHERE price > 10
GROUP BY type
ORDER BY AvgPrice DESC
```

В результирующем наборе группируются строки с одними и теми же значениями поля Type (при этом до начала группировки исключаются строки, не соответствующие условию WHERE). При группировке выполняется усреднение (AVG) столбца Price, а полученное среднее значение выставляется в результирующий набор в виде столбца AvgPrice. Значения столбца AvgPrice упорядочиваются по убыванию.

10. Исполнить оператор T-SQL и убедиться в правильности результата.

Создание таблицы для размещения результирующего набора

11. Ввести код T-SQL для создания таблицы TypeAvgPrice для значений результирующего набора, созданного по запросу п. 9:

```
USE pubs
SELECT type, AVG(price) AS AvgPrice
INTO TypeAvgPrice
FROM titles
WHERE price > 10
GROUP BY type
ORDER BY AvgPrice DESC
```

12. Исполнить запрос по ! и убедиться, что во вкладке 🗀***Результаты*** ничего не появилось, только во вкладке 🗀***Сообщения*** будет указано количество строк, на которое повлияло выполнение запроса.

13. Чтобы увидеть таблицу TypeAvgPrice, можно, например, исполнить запрос

```
SELECT * FROM TypeAvgPrice
```

(или просто в списке объектов выполнить ➢***Открыть таблицу*** в контекстном меню узла ***pubs → Таблицы →dbo.TypeAvgPrice***).

14. Ввести и выполнить запрос на уничтожение таблицы:

```
DROP TABLE TypeAvgPrice
```

§3.2. Упражнение «*Извлечение данных с помощью усложненных методик работы с запросами*»

Получение данных посредством внутреннего соединения

1. В окно редактора запросов ввести код T-SQL (можно использовать файл заготовок *Усложненные методики SELECT.txt*):

```
USE Northwind
SELECT o.CustomerID,o.OrderID,
s.CompanyName
FROM Orders o JOIN Shippers s
ON o.ShipVia = s.ShipperID
WHERE ShipCountry = 'USA'
ORDER BY o.CustomerID, s.CompanyName
```

Этот оператор SELECT извлекает идентификаторы покупателя, заказа и название компании-поставщика, ассоциированные с каждым *заказом* (база данных Northwind).

Поскольку название компании-поставщика указано в отдельной таблице (Shippers), необходимо *соединить* таблицы Orders и Shippers, используя в качестве условия соединения идентификатор поставщика. Столбец ShipVia является *внешним* ключом столбца ShipperID, и оба столбца содержат идентификаторы поставщиков.

Обратить внимание, что таблице Orders присвоен псевдоним **o**, а таблице Shippers – псевдоним **s**.

Замечание. В запросе использованы определенные умолчания:

- при описании соединения опущено ключевое слово **INNER**; строку без умолчания следовало бы писать как

```
FROM Orders o INNER JOIN Shippers s
```

• при введении псевдонимов опущены служебные слова **AS**; строку без умолчания следовало бы писать как

```
FROM Orders AS o JOIN Shippers AS s
```

2. Выполнить запрос по ! и проверить результат по вкладке 🗀***Результаты*** (предполагается результирующий набор 122 строки).

Получение данных посредством левого внешнего соединения

3. В окно редактора запросов ввести следующий код T-SQL:

```
USE Northwind
SELECT o.OrderID, o.CustomerID,
c.ContactName, c.City
FROM Orders o LEFT JOIN Customers c
ON o.CustomerID = c.CustomerID
AND o.ShipCity = c.City
ORDER BY o.OrderID
```

Здесь оператор SELECT используется для получения данных из столбцов OrderID, CustomerID (таблица Orders), а также ContactName, City (таблица Customers). Условие соединения использует столбцы CustomerID и *далее* равенство значений столбцов ShipCity и City.

4. Выполнить запрос по ! и проанализировать результаты.

Во вкладке 🗀***Сообщения*** указано, что запрос вернул 830 строк; так как указано левое внешнее соединение, то возвращаются *все* строки таблицы Orders (именно ее идентификатор стоит *слева* от конструкции LEFT JOIN), но из таблицы Customers возвращаются только те строки, где название города совпадает с тем, куда необходимо доставить заказ.

Во вкладке 🗀***Результаты*** «прокрутить» содержимое *до строки 108*.

Убедиться, что в столбцах ContactName и City указаны пустые значения (хотя если взглянуть на данные исходных таблиц, то можно заметить, что столбцы ContactName и City содержат значения). Дело в том, что хотя одно из условий соединения вы-

полняется (**o.CustomerID = c.CustomerID**), в столбец все равно помещаются пустые значения, поскольку второе условие (**o.ShipCity = c.City**) не выполняется, а в качестве связки условий соединения указано AND.

5. Для контроля переделать запрос с использованием *внутреннего* соединения:

```
FROM Orders o JOIN Customers c
```

Выполнить запрос и убедиться, что он возвращает 817 строк, так как из *обеих* таблиц возвращаются только строки, полностью соответствующие условию соединения.

Получение данных посредством правого внешнего соединения

6. Изменить в запросе п. 3 левое внешнее соединение на правое:

```
FROM Orders o RIGHT JOIN Customers c
```

7. Выполнить запрос по ! и проанализировать результаты.

Обратить внимание, что запрос вернул 820 строк; поскольку указано правое внешнее соединение, то возвращаются *все* строки таблицы Customers (именно ее идентификатор стоит *справа* от конструкции RIGHT JOIN), но из таблицы Orders возвращаются только те строки, где указан город, в котором живут покупатели.

По вкладке 🗀***Результаты*** убедиться, что первые три строки результирующего набора содержат пустые значения в столбцах OrderID и CustomerID, так как эти строки не соответствуют условию для таблицы Orders (**o.ShipCity = c.City**). Легко видеть, что 820-3=817 – это как раз те строки, которые дало внутреннее соединение в п. 5.

Получение данных посредством полного внешнего соединения

8. Изменить в запросе п. 6 правое внешнее соединение на полное:

```
FROM Orders o FULL JOIN Customers c
```

9. Выполнить запрос по ! и убедиться, что запрос вернул 833 строки; поскольку используется полное внешнее соединение, то возвращаются *все* строки из обеих таблиц.

Перейти во вкладку 🗀***Результаты*** и обратить внимание, что в первых трех строках результирующего набора в столбцах OrderID и CustomerID снова указываются пустые значения, а дальше со строки 111 (3+108) эти столбцы опять содержат пустые значения, как в п. 4.

Использование в подзапросе ключевого слова IN

10. В редакторе запросов ввести следующий код T-SQL:

```
USE Northwind
SELECT OrderID, EmployeeID AS EmpID
FROM Orders
WHERE EmployeeID IN
(
 SELECT EmployeeID
 FROM Employees
 WHERE City = 'Seattle'
)
ORDER BY OrderID
```

В этом операторе для определения работников, живущих в Сиэтле, применяется подзапрос; значения, которые он возвращает, затем используются в конструкции WHERE внешнего оператора SELECT для ограничения результирующего набора теми заказами, которые обработаны сотрудниками из Сиэтла.

11. Выполнить запрос и проанализировать результаты (должно получиться 227 строк).

Применение подзапроса с операторами сравнения и ключевым словом ALL (или ANY)

12. В редакторе запросов ввести следующий код T-SQL:

```
USE Northwind
SELECT OrderID, UnitPrice
FROM [Order Details]
WHERE UnitPrice > ALL
(
```

```
 SELECT UnitPrice
 FROM [Order Details] JOIN Orders
 ON [Order Details].OrderID=Orders.OrderID
 AND Orders.EmployeeID = 5
)
ORDER BY UnitPrice, OrderID
```

Этот оператор с помощью выполнения подзапроса определяет *максимальную* (так как используется ключевое слово ALL) цену заказов, обработанных сотрудником, которому присвоен номер 5 (можно убедиться выполнением *подзапроса*, что такая максимальная цена равна 210.80 долл.). Далее найденная цена используется в конструкции WHERE внешнего оператора SELECT для ограничения результирующего набора теми заказами, цена которых превышает сумму, определенную с помощью подзапроса.

Если вместо ключевого слова ALL использовать ANY, то подзапрос будет определять *минимальную* цену заказов, обработанных сотрудником с номером 5.

13. Исполнить запрос и проанализировать результаты (должно получиться 16 строк).

Использование подзапроса с ключевым словом EXISTS

14. Ввести следующий код T-SQL:

```
USE Northwind
SELECT OrderID, CustomerID
FROM Orders
WHERE EXISTS
(
SELECT * FROM Customers
WHERE Customers.CustomerID=Orders.CustomerID
AND City = 'London'
)
ORDER BY OrderID
```

Этот оператор с помощью выполнения подзапроса определяет покупателей из Лондона, для которых возвращается TRUE. Далее найденные строки используются в конструкции

WHERE внешнего оператора SELECT, чтобы определить заказы покупателей из Лондона.

15. Исполнить запрос и проанализировать результаты (должно получиться 46 строк).

Создание сводных данных посредством оператора ROLLUP (или CUBE)

16. Ввести следующий код T-SQL:

```
USE Northwind
SELECT ProductID, UnitPrice, SUM(Quantity)
AS 'Sum'
FROM [Order Details]
GROUP BY ProductID, UnitPrice
WITH ROLLUP
ORDER BY ProductID
```

Здесь оператор SELECT суммирует число проданных товаров (продуктов), а также выводит идентификаторы товаров и их цены за единицу.

17. Исполнить запрос и проанализировать результаты.

Обратить внимание, что в первой строке результирующего набора в полях ProductID и UnitPrice находятся пустые значения, а значение поля Sum представляет суммарное число единиц результирующего набора (иными словами, общее число *всех* проданных товаров). Остальные строки сначала группируются по товару, а затем строки внутри каждой группы по товару объединяются в подгруппы на основе цены.

Далее, в группе для каждого продукта присутствует строка, которая содержит пустое значение в поле UnitPrice, зато значение поля Sum в этой строке равно суммарному количеству проданных продуктов этого типа. Например, продано 174 единицы продукта 1 по цене 14.40 долл. за единицу и 654 единицы по цене 18.80 долл. за единицу. В итоге всего продано 828 единиц продукта с идентификатором (значением поля ProductID), равным 1.

«Прокрутить» результирующий набор (234 строки) и просмотреть итоговые значения для остальных продуктов.

При переходе от конструкции **WITH ROLLUP** к конструкции **WITH CUBE** в результирующий набор добавляется (в начале) еще 116 строк, представляющих еще один «информационный срез» – по полю UnitPrice (конкретно, сколько продано продуктов по цене 2.00 долл. за единицу, сколько по цене 2.50 долл. за единицу и т.д.). Вообще, конструкция **WITH CUBE** будет образовывать столько «информационных срезов», сколько полей будет указано в условии группировки.

§3.3. Упражнение *«Модификация данных в БД SQL Server»*

Создание тестовой таблицы в базе данных BookShopDB

1. В окне редактора запросов ввести код T-SQL:

```
USE BookShopDB
CREATE TABLE Test1
(
 RowID INT IDENTITY(1,1) NOT NULL,
 Title VARCHAR(80) NOT NULL,
 [Type] CHAR(12) NOT NULL DEFAULT ('Unknown'),
 City VARCHAR(60) NULL,
 Cost MONEY NULL
)
```

Заготовка кода находится в файле *Модификация данных.txt*.

2. Исполнить запрос и добиться сообщения об успешном выполнении.

Добавление к таблице Test1 данных с помощью оператора INSERT… VALUES

3. Ввести код T-SQL:

```
INSERT INTO Test1 (Title, [Type], Cost)
VALUES ('Test Title', 'business', 27.00)
```

4. Исполнить запрос и убедиться, что внеслась строка в таблицу (это можно сделать по **SELECT * FROM Test1** либо по ➢***Открыть таблицу*** контекстного меню таблицы).

Обратить внимание, что поле RowID сгенерировано *автоматически*, а поле City имеет пустое значение, так как ничего для него не было определено.

Добавление к таблице Test1 данных с помощью оператора INSERT… SELECT

5. Ввести следующий код T-SQL:

```
INSERT INTO Test1 (Title, [Type], Cost)
SELECT title, type, price
FROM pubs.dbo.titles
```

Этот оператор берет данные из таблицы Titles (база данных Pubs) и вставляет их в таблицу Test1 (18 строк).

6. Исполнить запрос и убедиться, что нужное количество строк вставлено в таблицу Test1, причем значения поля RowID сгенерированы автоматически, а в поле City каждой строки содержится пустое значение.

Модификация данных с помощью оператора UPDATE

7. Посмотреть содержимое таблицы Test1 (по **`SELECT * FROM Test1`** либо по ➢***Открыть таблицу*** контекстного меню таблицы) и запомнить названия нескольких книг, значение поля Type которых равно business, а также их цену (для последующей модификации).

8. Ввести код T-SQL:

```
UPDATE Test1
SET Cost = Cost * 2
WHERE [Type] = 'business'
```

Этот оператор в два раза увеличивает значение поля Cost по сравнению с исходным значением для книг типа business.

9. Исполнить запрос и сравнить значения полей Cost (запомненные в п. 7) для книг типа business.

Удаление данных из таблицы Test1 с помощью оператора DELETE

10. Ввести код T-SQL:

```
DELETE Test1
WHERE Title = 'Test Title'
```

Оператор удаляет из таблицы Test1 все строки, в столбце Title которых указано Test Title (видимо, одну строку, которая была внесена в п. 3).

11. Исполнить запрос и убедиться, что удалена одна строка с заголовком Test Title.

12. Очистить таблицу от данных по оператору

```
DELETE Test1
```

13. Удалить очищенную таблицу по оператору

```
DROP TABLE Test1
```

§ 4. ПРАКТИЧЕСКОЕ ЗАНЯТИЕ 4
Управление и манипулирование данными

В занятии рассматривается широкий круг вопросов, касающихся взаимодействия SQL Server с внешними источниками данных:

- импорт и экспорт данных (утилита bcp, оператор BULK INSERT) – упражнение *«Импорт и экспорт данных»* (§ 4.1);
- импорт и экспорт таблиц (технология ODBC взаимодействия Microsoft Access и Microsoft SQL Server) – упражнение *«Создание новой БД, экспорт и импорт таблиц (ODBC)»* (§ 4.2);
- доступ к внешним источникам данных (Microsoft Access) с помощью распределенных запросов к связанному серверу в рамках технологии OLE DB (операторы OPENQUERY и OPENROWSET, четырехкомпонентные имена) – упражнение *«Применение распределенных запросов для доступа к внешним данным»* (§ 4.3);
- извлечение данных с помощью курсора (конкретно, серверного курсора T-SQL) – упражнение *«Создание курсора для извлечения данных»* (§ 4.4).

§4.1. Упражнение *«Импорт и экспорт данных»*

Импорт данных в таблицу BookCondition (база данных BookShopDB) с помощью утилиты bcp

1. Скопировать файл *bookcondition.txt* в доступное место диска (далее предполагается, что это корневой каталог C:\).

2. Вызвать окно командной строки, например, так:

❑***Пуск*** ➢***Выполнить...***

{далее в окне ☐ ***Запуск программы***}

Открыть := cmd ❑***OK***

3. Установить нужный каталог (тот, где находится скопированный в п. 1 файл), например, командой **`cd c:\`**

4. В командной строке набрать

```
bcp BookShopDB..BookCondition in bookcondition.txt -c -T
```

Параметр **-c** задает символьный тип данных, а **-T** указывает, что надо использовать доверенное соединение с сервером.

Замечание. При вводе команды следует набирать текст *абсолютно точно*, с точностью до символа. В частности, при вводе параметра **-c** надо использовать строчную букву, а при вводе **-T** – прописную.

По нажатию [Enter] произойдет копирование данных из файла *bookcondition.txt* в таблицу BookCondition; по завершению копирования выдается сообщение с числом скопированных строк, размером сетевого пакета и временем выполнения запроса.

5. Просмотреть содержимое таблицы BookCondition (по **SELECT * FROM BookCondition** либо по ➢***Открыть таблицу*** контекстного меню таблицы).

Обратить внимание, что в каждой строке столбца Description стоит подробное описание оценки состояния книги; при отсутствии такого описания в этой строке стояло бы значение по умолчанию N/A (это значение можно увидеть в свойствах столбца Description, доступных по пункту ➢***Свойства*** контекстного меню узла дерева объектов ***Таблицы → BookCondition → Столбцы → Description***).

Импорт данных в таблицу Positions (база данных BookShopDB) с помощью оператора BULK INSERT

6. Скопировать файл *positions.txt* в доступное место диска (далее предполагается, что это корневой каталог C:\).

7. Ввести в окно редактора запросов следующий код T-SQL:

```
USE BookShopDB
BULK INSERT Positions
FROM 'c:\positions.txt'
WITH (DATAFILETYPE = 'CHAR')
```

Текстовую заготовку данного запроса можно скопировать из файла *Импорт данных.txt*.

8. Выполнить запрос и убедиться, что таблица Positions заполнена; в частности, у каждой должности имеется значение идентификатора PositionID**.**

Экспорт данных в текстовый файл с помощью утилиты bcp

9. В командной строке (см. пп. 2–4) набрать

```
bcp BookShopDB..Positions out Positions2.txt –c –T
```

Утилита должна *создать* текстовый файл *Positions2.txt* в указанном каталоге и скопировать туда (**out**) данные из таблицы Positions базы данных BookShopDB. По завершению копирования выдается сообщение о числе скопированных строк, размере сетевого пакета и затраченном на исполнение запроса времени.

10. Просмотреть файл *Positions2.txt* в Блокноте и убедиться, что каждая строка таблицы находится в отдельной строке файла, а значения полей разделены символами табуляции.

§ 4.2. Упражнение «*Создание новой БД, экспорт и импорт таблиц (ODBC)*»

Постановка задачи. Создать в Microsoft SQL Server новую пользовательскую базу данных SQL2005new. Новая пользовательская БД должна содержать *все* таблицы, включенные из базы данных *study5_08.mdb* Microsoft Access. Заполнение БД Microsoft SQL Server выполнить с помощью импорта/экспорта таблиц, для чего создать новый системный источник ODBC (если это возможно в данной версии Microsoft SQL Server).

Находясь в базе данных SQL2005new на Microsoft SQL Server, осуществить *импорт* таблиц Группы и Слушатели из базы данных *study5_08.mdb* Microsoft Access.

Находясь в базе данных Microsoft Access *study5_08.mdb*, осуществить *экспорт* таблиц Преподаватели, Должности и Слушатели в базу данных SQL2005new на Microsoft SQL Server.

1. Убедиться, что Microsoft SQL Server запущен (в Панели задач соответствующего значка может и не быть, но при старте Management Studio выдается ошибка соединения).

Для запуска активизировать в меню «Пуск»:

❑***Пуск*** ➢***Программы*** ➢ ***Microsoft SQL Server 2005*** ➢***Средства настройки*** ➢ ***SQL Server Configuration Manager***

и в Диспетчере конфигурации выполнить:

⇗ Службы SQL Server 2005 ⇗ SQL Server (MSSQLSERVER) ❑▶

2. Установить связь с SQL Server через Management Studio; использовать имя сервера по умолчанию и следующие установки проверки подлинности:

Проверка подлинности: ⇩Проверка подлинности Windows
❑Соединить

3. В *контекстном* меню узла ***Базы данных*** выполнить ***➢Создать базу данных…***

В диалоговом окне ☐ ***Создание базы данных*** оставить все параметры по умолчанию, только задать имя базы данных (SQL2005new) и нажать ❑***ОК.***

Убедиться, что база данных создана в полном составе: ***Диаграммы – Таблицы – Представления – Синонимы – Программирование – Компонент Service Broker – Хранилище – Безопасность***. По умолчанию физическое расположение БД следующее:

C:\Program Files\MS SQL Server\MSSQL.1\MSSQL\Data

4. Импортировать таблицы Группы и Слушатели через *контекстное* меню базы данных SQL2005new: ***➢Задачи ➢Импорт данных…***

К сожалению, Мастер импорта и экспорта SQL Server *не работает* (как в предыдущих версиях) с источником данных вида ***⇩Other (ODBC Data Source)***. Если бы этот тип источника поддерживался, то далее следовало бы делать системный источник данных ODBC по ❑***New…*** и далее «подвешивать» к нему Microsoft Access Driver.

В настоящей версии SQL Server приходится в качестве источника данных выбирать ***⇩Microsoft Access*** и далее указывать имя файла (*study5_08.mdb*) через ❑***Обзор…***

В качестве назначения импорта *ничего другого не остается делать*, как выбирать ***⇩Microsoft OLE DB Provider for SQL Server***. Тогда Мастер импорта и экспорта «подкинет» имя сервера, указание проверки подлинности (Windows) и имя базы данных (SQL2005new).

После нажатии ❑***Далее>*** нужно установить ⊙***Скопировать данные из одной или нескольких таблиц или представлений*** , тогда при нажатии ❑***Далее>*** появится список таблиц и пред-

ставлений, из которого нужно выбрать таблицы Группы и Слушатели.

Замечание. Префиксы [dbo] добавляются в SQL Server *автоматически*; кнопкой ❑***Предварительный просмотр*** можно просматривать выделенную таблицу.

Работа Мастера завершается *немедленным* выполнением пакета (в смысле *SSIS* – SQL Server Integration Services) создания таблиц и копирования данных.

Убедиться, что таблицы Группы и Слушатели (в виде dbo.Группы и dbo.Слушатели) появились в базе данных SQL2005new, посмотреть их из контекстного меню ➢***Открыть таблицу…*** Убедиться, что первичный ключ для поля Код_студента *исчез*.

5. Экспортировать таблицы Преподаватели, Должности и Кафедры из базы данных *study5_08.mdb* в базу данных SQL2005new. Этот экспорт надо делать *по очереди* с каждой таблицей.

Например, открыть в Microsoft Access базу данных *study5_08.mdb* и выбрать таблицу Преподаватели. В ленте выбрать ➢***Работа с базами данных*** ➢***Перемещение данных*** ❑***SQL Server.*** В результате стартует Мастер преобразования в формат SQL Server (в котором технология ODBC работает в полном объеме).

Следует указать ⊙***Использовать существующую базу данных***, в результате чего откроется окно ❐***Выбор источника данных.*** Нужно работать с вкладкой 🗀***Источник данных компьютера***, в результате чего откроется большой список типов имеющихся *внешних* источников данных (dBase, Microsoft Excel, Microsoft Access, Microsoft Visio и т.п.). Тем не менее, выбирать что-либо из этого списка *не нужно*, а следует нажать ❑***Создать…*** и создавать источник данных ⊙***системный (только для этого компьютера)***. В результате открывается список ODBC-драйверов, в котором нужно выбрать ⇩ ***SQL Server*** (самый последний) (рис. 4).

Далее нужно ввести имя источника (например, access1) и описание источника (например, Экспорт из Access в SQL Server). В качестве имени сервера из списка выбрать имя по

умолчанию. Перед пересылкой неплохо *проверить* источник данных ODBC по ❑***Проверить источник данных***.

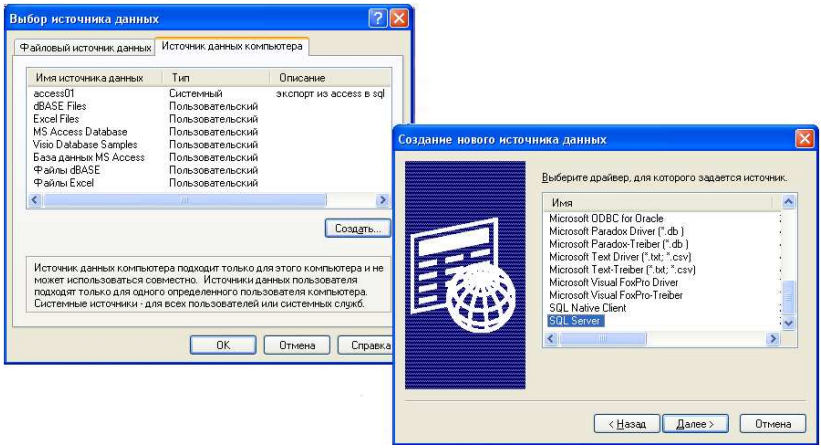


Рис. 4 — Создание источника данных ODBC

Замечания.

• Следует иметь в виду, что при работе Мастера преобразования в формат SQL Server устанавливается база данных *по умолчанию* (то есть БД master). Однако перемещение всех таблиц из БД master в БД SQL2005new практически невозможно, поэтому *обязательно* нужно при работе Мастера по созданию источника данных для SQL Server установить:

☑ ***Использовать по умолчанию базу данных***
⇩***SQL2005new***

• Примерно такой же процесс можно запустить через *контекстное* меню соответствующей таблицы в Microsoft Access: ➢***Экспорт*** ➢***База данных ODBC***. В этом случае уж точно нужно работать с каждой таблицей по очереди.

• Для того, чтобы увидеть пересланные таблицы, надо в SQL Server в *контекстном* меню соответствующей базы данных (в данном случае, SQL2005new) выполнить ➢***Обновить.***

6. *Факультативно*. Создать первичные ключи для таблиц Кафедры (Код_кафедры), Преподаватели (Код_преподавателя), Группы (Группа, Форма_обучения, Год_приема), Должности (Код_должности), Слушатели (Код_студента).

Для этого выделить таблицу и в *контекстном* меню выбрать ➢***Изменить***, а далее в схеме данных в контекстном меню строки выполнить ➢***Задать первичный ключ.***

Замечание. Задание первичного ключа *на сервере* не всегда удачно; удобнее это делать в связанной таблице в Microsoft Access. За счет этого пользователи могут по-разному строить ключи в таблицах и, таким образом, организовывать разные схемы данных.

§4.3. Упражнение *«Применение распределенных запросов для доступа к внешним данным»*

Определение связанного сервера

1. Скопировать базу данных *study5_08.mdb* Microsoft Access в доступную область диска (ниже предполагается, что это корневой каталог диска C:\).

2. Открыть Management Studio и в контекстном меню узла ***Объекты сервера → Связанные серверы*** выполнить ➢***Создать связанный сервер…***

3. В окне ☐ ***Создание связанного сервера*** установить:

Тип сервера ⊙ ***Другой источник данных***
Связанный сервер := TEST_SERVER
Поставщик ⇩***Microsoft Jet 4.0 OLE DB Provider***
Название продукта := study
Источник данных := C:\study5_08.mdb
❑***OK***

Убедиться, что TEST_SERVER появился в дереве объектов как связанный сервер.

Выполнение запроса к внешнему источнику данных с помощью оператора OPENQUERY

4. В окне редактора запросов ввести код T-SQL (заготовки этого и последующих запросов содержатся в файле *Распределенные запросы.txt*):

```
SELECT *
FROM OPENQUERY
(TEST_SERVER, 'SELECT
```

```
Фамилия, Имя, Отчество FROM Преподаватели')
```

5. Выполнить запрос (в котором оператор OPENQUERY идентифицирует связанный сервер TEST_SERVER и определяет оператор SELECT, который возвращает значения столбцов Фамилия, Имя, Отчество из таблицы Преподаватели) и убедиться, что данные (7 строк) возвращены правильно.

Выполнение запроса к внешнему источнику данных с помощью оператора T-SQL

6. Ввести в окне редактора запросов следующий код T-SQL:

```
SELECT Фамилия, Имя, Отчество
FROM TEST_SERVER...Преподаватели
```

Обратить внимание на четырехкомпонентное имя, ссылающееся на объект связанного сервера (таблицу Преподаватели).

7. Выполнить запрос и убедиться, что результаты совпадают с полученными в п. 5.

Выполнение запроса к внешнему источнику данных с помощью функции OPENROWSET

8. Ввести в окне редактора запросов следующий код T-SQL:

```
SELECT Фамилия, Имя, Отчество
FROM OPENROWSET
('Microsoft.Jet.oledb.4.0','C:\study5_08.mdb';
'admin'; '', Преподаватели)
```

Замечание. Придется переустанавливать параметры сервера для работы с распределенными запросами (*до* выполнения кода п. 8):

```
EXEC sp_configure 'show advanced options', 1
RECONFIGURE
GO
EXEC sp_configure 'Ad Hoc Distributed Queries', 1
RECONFIGURE
GO
```

9. Выполнить запрос п. 8 (в нем указывается, что компонентом доступа (поставщиком) OLE DB является Microsoft.Jet.oledb.4.0, источником данных – C:\study5_08.mdb,

идентификатором пользователя – admin, пароль отсутствует) и убедиться, что результаты совпадают с полученными в п. 5.

§4.4. Упражнение
«Создание курсора для извлечения данных»

Объявление курсора с помощью оператора DECLARE CURSOR и его заполнение с помощью оператора OPEN

1. Ввести в окне редактора запросов следующий код T-SQL:

```
USE SQL2005new
DECLARE PrepCrs CURSOR FOR
 SELECT * FROM Преподаватели
 WHERE Уч_степень LIKE 'к.%'
 ORDER BY Фамилия, Имя, Отчество
```

(Заготовка данного запроса может быть взята из файла *Извлечение данных с помощью курсора.txt*).

Оператор объявляет курсор T-SQL с именем PrepCrs, связанный с оператором SELECT, извлекающим из таблицы Преподаватели всех преподавателей – кандидатов наук.

Замечание. Для объявления двустороннего курсора следует оператор DECLARE CURSOR использовать в виде

```
DECLARE PrepCrs SCROLL CURSOR FOR
```

2. Исполнить оператор и добиться сообщения об успешном выполнении команды.

3. Ввести в окне редактора запросов следующий оператор T-SQL:

```
OPEN PrepCrs
```

Этот оператор заполняет курсор результирующим набором оператора SELECT, заданного в п. 1.

4. Исполнить оператор и добиться сообщения об успешном завершении команды.

Извлечение из курсора строки по оператору FETCH

5. Ввести следующий код T-SQL:

```
FETCH NEXT FROM PrepCrs
```

Оператор извлекает *следующую* строку из результирующего набора. Так как эта операция – первая операция извлече-

ния в сеансе, то возвращается *первая* строка результирующего набора (можно сказать, что изначально курсор указывает *нулевую* строку этого набора).

6. Исполнить оператор, введенный в п. 5, и увидеть во вкладке 🗀***Результаты*** первую строку результирующего набора.

7. Исполнить оператор п. 5 еще раз и убедиться, что выводится вторая строка результирующего набора.

Закрытие курсора с помощью оператора CLOSE и его освобождение с помощью оператора DEALLOCATE

8. Ввести и выполнить оператор T-SQL:

```
CLOSE PrepCrs
```

Появится сообщение об успешном завершении команды; таким образом, курсор закрыт.

9. Ввести и выполнить оператор T-SQL:

```
DEALLOCATE PrepCrs
```

Появится сообщение об успешном завершении команды; таким образом, курсор освобожден.

§ 5. ПРАКТИЧЕСКОЕ ЗАНЯТИЕ 5
Хранимые процедуры

Занятие начинается с изучения принципов работы с *готовыми* процедурами и просмотра их содержимого (упражнение *«Изучение хранимых процедур»* (§ 5.1));

Упражнение *«Работа с хранимыми процедурами»* (§ 5.2) посвящено созданию, просмотру зависимостей, исполнению, модификации, удалению хранимой процедуры.

Далее естественно рассмотреть особенности программирования хранимых процедур (входные и выходные параметры, алгоритмы обработки ошибок, вложенные хранимые процедуры, отладка хранимых процедур) – упражнение *«Программирование хранимой процедуры для добавления и извлечения данных»* (§ 5.3).

Занятие заканчивается «полуавтоматическим» созданием хранимой процедуры (из запроса или представления) с использованием графического интерфейса – упражнение *«Создание хранимых процедур в Management Studio»* (§ 5.4).

§5.1. Упражнение *«Изучение хранимых процедур»*

Просмотр системных хранимых процедур в базе данных Master

1. В обозревателе объектов Management Studio открыть узел ***Master*** и далее узел ***Программирование → Хранимые процедуры*** (обратить внимание, что есть еще узел ***Расширенные хранимые процедуры***).

2. Раскрыть узел ***Хранимые процедуры → Системные хранимые процедуры.*** Обратить внимание, что у процедур два владельца – sys и dbo. Можно также раскрыть узел ***Системные расширенные хранимые процедуры*** и убедиться, что у них единственный владелец – sys.

3. Раскрыть узел, соответствующий системной хранимой процедуре ***sys.sp_who*** и далее узел ***Параметры***. По этому узлу видно, что входным параметром процедуры является

@loginame – типа nvarchar (128), значения по умолчанию нет; выходным параметром является целое значение (видимо, код возврата).

4. Убедиться, что последние четыре системные процедуры с префиксом xp_ (sys.xp_grantlogin, sys.xp_logininfo, sys.xp_repl_convert_encrypt_sysadmin_wrapper, sys.xp_revokelogin) на самом деле не являются расширенными процедурами. Это можно сделать, например, так:

```
USE Master
SELECT OBJECTPROPERTY (object_id('xp_logininfo'),
     'IsExtendedProc')
```

После выполнения этого запроса возвращается значение 0.

Два метода просмотра содержимого хранимой процедуры

5. Вызвать правой кнопкой мыши контекстное меню процедуры sys.sp_who и выполнить пункт ➢***Изменить.*** В окне редактора запросов появится полный размеченный текст процедуры.

6. В окне редактора запросов ввести и выполнить следующий код T-SQL:

```
sp_helptext [master.sys.sp_who]
```

В итоге содержимое системной хранимой процедуры sp_who выведется на панели 🗀***Результаты.***

§5.2. Упражнение «*Работа с хранимыми процедурами*»

Создание хранимой процедуры в базе данных Northwind

1. В окне редактора запросов ввести код T-SQL:

```
USE Northwind
GO
CREATE PROCEDURE dbo.CustOrderHistRep
@CustomerID char(5)
AS
SELECT ContactName, ContactTitle
FROM Customers WHERE CustomerID = @CustomerID
SELECT ProductName, Total=SUM(Quantity)
FROM Products P, [Order Details] OD, Orders O,
     Customers C
WHERE
```

```
      C.CustomerID = @CustomerID AND
      C.CustomerID = O.CustomerID AND
      O.OrderID = OD.OrderID AND
      OD.ProductID = P.ProductID
GROUP BY ProductName
GO
```

Этот текст (как и текст п. 11) удобно взять из файла *Работа с хранимыми процедурами.txt*.

Здесь длина параметра @CustomerID равна 5 символам, так как если выполнить запрос к таблице [northwind].[dbo].[customers], то видно, что длина всех идентификаторов покупателей равна 5 символов.

Замечание. *Единственное* ключевое слово AS в данном случае *разделяет* заголовок процедуры (в таблице SysObjects) и текст процедуры (в таблице SysComments).

Следует напомнить, что все остальные использования ключевого слова AS относятся к введению имен-псевдонимов (например,

```
FROM Products AS P, [Order Details] AS OD,
     Orders AS O, Customers AS C                    )
```

и по умолчанию опускаются.

2. Просмотреть текст процедуры (коротко) и понять, что при ее исполнении (когда она уже будет создана) сначала выведется имя контактного лица и заголовок контактной информации (первый SELECT), а далее названия и общее количество (SUM) единиц каждого товара, который приобрел покупатель (второй SELECT). Результирующий набор группируется по названию товара.

Следует заметить, что соединения таблиц реализованы в конструкции WHERE, а не в конструкции FROM. При модификации процедуры (п. 11) соединения (с помощью выражения JOIN) перемещаются в конструкцию FROM.

Просмотр хранимых процедур в анализаторе запросов

3. Выполнить запрос п. 1 и убедиться, что процедура dbo.CustOrderHistRep появилась в списке пользовательских процедур базы данных Northwind (узел ***Программирование***). Текст процедуры всегда доступен по контекстному меню ***➢Изменить.***

4. Открыть узел ***Параметры*** хранимой процедуры dbo.CustOrderHistRep. Убедиться, что у процедуры два параметра – @CustomerID и встроенный параметр для хранения кода возврата @RETURN_VALUE (в некоторых версиях SQL Server – просто фраза «Возвращает целое значение»).

5. Посмотреть зависимости данной хранимой процедуры по контекстному меню ➢***Просмотреть зависимости***. Убедиться, что процедура зависит от четырех объектов (таблиц Orders, Products, Order Details, Customers из базы данных Northwind), а от самой хранимой процедуры не зависит ни один объект.

6. Ту же работу выполнить с помощью системной хранимой процедуры sp_depends. Для этого открыть новый запрос по ❑***Создать запрос***, ввести и выполнить следующую команду:

```
sp_depends CustOrderHistRep
```

Замечание. В поле Name при выполнении запроса будут повторяющиеся значения, но все записи с повторяющимися элементами отличаются значениями *других* полей.

Исполнение хранимой процедуры

7. В том же окне редактора запросов ниже ввести и исполнить команду

```
EXEC [northwind].[dbo].[custorderhistrep]
     @CustomerID='THECR'
```

Замечания.

• Команда обращения к процедуре специально в этом случае вводится так, что отсутствуют ограничения по использованию верхнего и нижнего регистра. См. замечание к п. 1 § 3.1.

• В данном случае вводится полное имя процедуры, что в принципе не обязательно, но позволяет не делать активной базу данных Northwind.

В результате должно возвратиться *два* результирующих набора: первый (имя контактного лица и заголовок контактной информации) – в верхней части вкладки 🗀***Результаты***, второй (название и количество товара) – в нижней части вкладки 🗀***Результаты.***

8. Попробовать второй способ исполнения хранимой процедуры – *через сценарий*. Для этого в контекстном меню хранимой процедуры выполнить ➢***Создать сценарий для хранимой***

процедуры ➢Используя EXECUTE ➢В новом окне редактора запросов.

По тексту сценария видно, что объявляются две переменные – @RC (код возврата – Return Code) и @CustomerID, при этом в операторе EXEC @RC соответствует хранимой процедуре (через знак =).

9. В окне редактора запросов щелкнуть мышью после слова @CustomerID и дописать в строку символы ='THECR'. Таким образом, оператор EXEC должен принять вид

```
EXEC @RC=[Northwind].[dbo].[CustOrderHistRep]
     @CustomerID='THECR'
```

10. Исполнить запрос и убедиться в том, что результирующие наборы такие же, как в п. 7.

Модификация хранимой процедуры

11. В окне редактора запросов ввести следующий код T-SQL:

```
USE Northwind
GO
ALTER PROCEDURE dbo.CustOrderHistRep
@CustomerID char(5)
AS
SELECT ContactName, ContactTitle
FROM Customers WHERE CustomerID = @CustomerID
SELECT ProductName, Total=SUM(Quantity)
FROM Products P INNER JOIN [Order Details] OD
ON P.ProductID = OD.ProductID JOIN Orders O
ON OD.OrderID = O.OrderID JOIN Customers C
ON O.CustomerID = C.CustomerID
WHERE C.CustomerID = @CustomerID
GROUP BY ProductName
ORDER BY Total DESC
GO
```

12. Исполнить запрос, а далее для того, чтобы убедиться, что текст процедуры изменился, исполнить оператор

```
sp_helptext CustOrderHistRep
```

Во вкладке 🗀***Результаты*** появится текст хранимой процедуры.

13. Убедиться, что результат измененной процедуры несколько отличается (прежде всего, упорядочиванием) от резуль-

татов пп. 7 и 9. Можно это просто сделать по контекстному меню хранимой процедуры ➢***Выполнить хранимую процедуру…***

Появится окно выполнения процедуры, в котором надо для параметра @CustomerID ввести значение THECR (без апострофов).

Удаление хранимой процедуры

14. В нижней части окна редактора запросов ввести и исполнить оператор

```
DROP PROCEDURE dbo.CustOrderHistRep
```

При этом база данных Northwind должна быть установлена в качестве текущей (можно по списку баз данных).

15. Убедиться (например, по списку в обозревателе объектов), что хранимая процедура удалена.

Следует заметить, что удаление хранимой процедуры может быть выполнено из контекстного меню ➢***Удалить*** этой процедуры.

§5.3. Упражнение *«Программирование хранимой процедуры для добавления и извлечения данных»*

Данное упражнение объединяет многие аспекты программирования хранимых процедур:

- создание процедуры для проверки наличия в БД BookShopDB повторяющихся записей о покупателях;
- включение этой процедуры в процедуру добавления новых записей в БД BookShopDB;
- использование переменных, параметров, кода возврата;
- использование системной переменной @@ERROR и управление ходом выполнения.

Создание пользовательской хранимой процедуры

1. В окне редактора запросов ввести код T-SQL (заготовки этого и последующих запросов содержатся в файле *Программирование хранимой процедуры.txt*):

```
USE BookShopDB
GO
CREATE PROCEDURE dbo.AddCustomer
```

```
-- CustomerID не входит в параметры, поскольку
-- идентификатор генерируется автоматически
-- (в столбце с идентификатором)
@FirstName varchar(30), @LastName varchar(30),
@Phone char(15), @Address1 varchar(30),
@Address2 varchar(30)='unknown', @City varchar(20),
@State char(2), @Zip char(6)
AS
INSERT [BookShopDB].[dbo].[Customers]
(FirstName, LastName, Phone, Address1,
Address2, City, State, Zip)
VALUES
(@FirstName, @LastName, @Phone, @Address1,
@Address2, @City, @State, @Zip)
RETURN(SELECT @@IDENTITY AS 'Identity')
GO
```

Замечание. По умолчанию для параметра @Address2, который не является обязательным, принято значение «unknown» (ограничение CHECK). В операторе RETURN используется функция @@IDENTITY, возвращающая значение идентификатора пользователя.

2. Исполнить код п. 1 и убедиться через обозреватель объектов, что хранимая процедура AddCustomer создана.

3. В окне редактора запросов ввести и выполнить код обращения к процедуре:

```
DECLARE @r_Code int
EXECUTE @r_Code = dbo.AddCustomer
@FirstName = 'Jamie', @LastName = 'Marra',
@Phone ='425-555-1212',@Address1 ='20 Oak St., SE',
@City = 'Remy', @State = 'WA' , @Zip = '98888'
SELECT [CustomerID] = 'The new customer ID is:' +
CONVERT(CHAR(5), @r_Code)
```

Таким образом, при исполнении процедуры код возврата сохранит значение функции @@IDENTITY, которое и выведется после обращения к процедуре.

Добавление к хранимой процедуре алгоритма обработки ошибок

4. Следует модифицировать процедуру AddCustomer, добавив команды обработки ошибок, примерно так:

```
     ALTER PROCEDURE dbo.AddCustomer
@FirstName varchar(30)= 'unknown',
@LastName varchar(30)='unknown',
@Phone char(15)= NULL,
@Address1 varchar(30) = NULL,
@Address2 varchar(30) = 'unknown',
@City varchar(20) = NULL,
@State char(2) = NULL,
@Zip char(6) = NULL
     AS
IF (@FirstName='unknown') AND (@LastName='unknown')
     RETURN(1)
     ELSE IF @Phone IS NULL
     RETURN(2)
     ELSE IF
     @Address1 IS NULL OR @City IS NULL OR
     @State IS NULL OR @Zip IS NULL
     RETURN(3)
     ELSE
     INSERT [BookShopDB].[dbo].[Customers]
     (FirstName, LastName, Phone, Address1,
     Address2, City, State, Zip)
     VALUES
     (@FirstName, @LastName, @Phone, @Address1,
     @Address2, @City, @State, @Zip)
     RETURN(SELECT @@IDENTITY AS 'Identity')
     IF @@ERROR <> 0
     RETURN(4)
     GO
```

Можно заметить, что для каждой переменной задано значение по умолчанию; таким образом можно избежать ошибок. Например, если не ввести значение параметра @FirstName, оно по умолчанию примется равным «unknown»; в результате процедура выполнится корректно, но вернет код возврата 1. Это не противоречит бизнес-правилу, согласно которому необходимо ввести имя или фамилию, и соответствует ограничению CHECK, налагаемому на поля FirstName и LastName в базе данных BookShopDB.

Аналогично, если не введен номер телефона, по умолчанию используется значение NULL, а исполняемой процедурой посылается код возврата 2; если отсутствует какая-то часть ад-

реса, то посылается код возврата 3 (параметр @Address2 не задействован в процедуре проверки адреса, так как он не обязателен).

Если заданы все параметры, то вызывается оператор INSERT, но если его исполнение заканчивается неудачей из-за ошибки БД, исполняемой процедурой посылается код возврата, равный 4.

По идее, можно еще добавлять команды для проверки (обработки) ошибок; например, перед созданием новой записи о покупателе следует проверить наличие дублирующих сведений о том же покупателе.

5. Для вызова процедуры в исправленном виде ввести и исполнить следующий код T-SQL:

```
     DECLARE @r_Code int
     EXECUTE @r_Code = dbo.AddCustomer
@FirstName= 'Jamie', @LastName = 'Marra',
@Phone= '425-555-1212', @Address1='20 Oak St., SE',
@City = 'Remy' , @State = 'WA' , @Zip = '98888'
     IF @r_Code = 4
     BEGIN
     PRINT 'A database error has occured.
     Please contact the help desk for assistance.'
     END
     IF @r_Code = 1
     PRINT 'You must specify a value for the
     firstname or lastname'
     ELSE IF @r_Code = 2
     PRINT 'You must specify a value for the phone
     number'
     ELSE IF @r_Code = 3
     PRINT 'You must provide all address
     information: Street address, City, State
     and Zipcode'
     ELSE IF @r_Code = @@IDENTITY
     SELECT [Customer ID] = 'The new customer ID
      is: ' + CONVERT (CHAR(5), @r_Code)
```

Замечание. В примере намеренно вводится та же информация о покупателе, что и при предыдущем выполнении процедуры (п. 2). Это наталкивает на мысль о создании дополнитель-

ной процедуры (вложенной), которая проверяет, создается ли повторяющаяся запись о покупателях.

Создание хранимой процедуры, защищающей от дублирования информации о покупателях

6. Ввести в окне редактора запросов и исполнить следующий код T-SQL:

```
    CREATE PROCEDURE dbo.CheckForDuplicateCustomer
@1_FirstName varchar(30)= 'unknown',
@1_LastName varchar(30)= 'unknown',
@1_City varchar(20) = NULL,
@1_State char(2) = NULL,
@1_Phone char(15) = NULL,
@o_FirstName varchar(30) OUTPUT,
@o_LastName varchar(30) OUTPUT,
@o_City varchar(20) OUTPUT,
@o_State char(2) OUTPUT,
@o_Phone char(15) OUTPUT
    AS
SELECT @o_FirstName=FirstName,@o_LastName=LastName,
      @o_City=City, @o_State=State, @o_Phone=Phone
    FROM Customers
    WHERE FirstName=@1_FirstName AND
    LastName=@1_LastName AND
    City=@1_City AND State=@1_State AND
    Phone=@1_Phone
    IF @@ROWCOUNT <> 0
    RETURN (5)
```

Эта процедура проверяет таблицу Customers на наличие в ней записей с одинаковыми именами, фамилиями, городами, штатами и телефонами. Имена *входных* параметров начинаются с «1», что позволяет их отличить от соответствующих параметров в процедуре AddCustomer.

Оператор SELECT устанавливает *выходные* параметры (начинающиеся с «o») равными значениям соответствующих столбцов таблицы Customers. Проверка на совпадение записей (по AND – полное совпадение) выполняется в конструкции WHERE (проверяется равенство столбцов и *входных* параметров).

Таким образом, если отобрано число совпадающих записей, большее нуля (то есть если совпадение выявлено), то функция @@ROWCOUNT возвращает значение больше нуля, поэтому устанавливается код возврата, равный 5.

7. Теперь надо *встроить* процедуру проверки внутрь основной процедуры, для чего ввести код T-SQL и исполнить его:

```
     ALTER PROCEDURE dbo.AddCustomer
@FirstName varchar(30)= 'unknown',
@LastName varchar(30)= 'unknown',
@Phone char(15) = NULL,
@Address1 varchar(30) = NULL,
@Address2 varchar(30) = 'unknown',
@City varchar(20) = NULL,
@State char(2) = NULL,
@Zip char(6) = NULL
     AS
IF (@FirstName='unknown') AND (@LastName='unknown')
     RETURN(1)
     ELSE IF @Phone IS NULL
     RETURN(2)
     ELSE IF @Address1 IS NULL OR @City IS NULL
     OR @State IS NULL OR @Zip IS NULL
     RETURN(3)
     -- Начало вложенной процедуры
     DECLARE @r_Code int, @v_FirstName varchar(30),
     @v_LastName varchar(30), @v_City varchar(20),
     @v_State char(2), @v_Phone char(15)
EXECUTE @r_Code = dbo.CheckForDuplicateCustomer
     @1_FirstName = @FirstName,
     @1_LastName = @LastName,
     @1_City = @City, @1_State = @State,
     @1_Phone = @Phone,
     @o_FirstName = @v_FirstName OUTPUT,
     @o_LastName = @v_LastName OUTPUT,
     @o_City = @v_City OUTPUT,
     @o_State = @v_State OUTPUT,
     @o_Phone = @v_Phone OUTPUT
     IF @@ROWCOUNT > 0
     BEGIN
     PRINT 'A duplicate record was found for ' +
     @v_FirstName + ' ' + @v_LastName
```

```
PRINT 'in ' + @v_City + ' ' + @v_State +
' with a phone number '
PRINT 'of ' + @v_Phone + '. '
RETURN(5)
END
-- Конец вложенной процедуры
INSERT [BookShopDB].[dbo].[Customers]
(FirstName, LastName, Phone,
Address1, Address2, City, State, Zip)
VALUES
(@FirstName, @LastName, @Phone,
@Address1, @Address2, @City, @State, @Zip)
RETURN(SELECT @@IDENTITY AS 'Identity')
IF @@ERROR <> 0
RETURN(4)
GO
```

Хранимая процедура CheckForDuplicateCustomer вложена в процедуру AddCustomer (см. комментарии в ее начале и конце). Переменные, имена которых начинаются с «v_», служат для хранения *выходных* параметров. *Входным* параметрам (имена которых начинаются с «1_») присваиваются значения соответствующих *входных* параметров процедуры AddCustomer, заданные при исполнении. Затем *выходным* параметрам (начинающимся с «v_») присваиваются значения соответствующих *выходных* параметров процедуры CheckForDuplicateCustomer и далее используются в операторах PRINT. Эти операторы выполняются, если функция @@ROWCOUNT вернула значение, отличное от нуля.

Проверка хранимых процедур

8. Ввести в окне редактора запросов и выполнить следующий код T-SQL для вызова и проверки процедур:

```
DECLARE @r_Code int
EXECUTE @r_Code = dbo.AddCustomer
@FirstName= 'Jamie', @LastName = 'Marra',
@Phone = '425-555-1212',
@Address1 = '20 Oak St., SE',
@City = 'Remy' , @State = 'WA' , @Zip = '98888'
IF @r_Code = 4
BEGIN
```

```
     PRINT 'A database error has occured.
     Please contact the help desk for assistance.'
     END
     IF @r_Code = 1
     PRINT 'You must specify a value for the
     firstname or lastname'
     ELSE IF @r_Code = 2
     PRINT 'You must specify a value for the phone
     number'
     ELSE IF @r_Code = 3
PRINT 'You must provide all address information:
Street address, City, State and Zipcode'
     ELSE IF @r_Code = @@IDENTITY
SELECT [Customer ID] = 'The new customer ID is: ' +
      CONVERT (CHAR(5), @r_Code)
```

В данном случае после выполнения кода должно быть сообщение о том, что запись уже существует:

```
A duplicate record was found for Jamie Marra
in Remy WA with a phone number of 425-555-1212.
```

9. Изменить значение параметра @FirstName на 'Jeff', а @LastName на 'Fellinge' и снова исполнить процедуру. В результате должно появиться сообщение о занесении нового покупателя:

```
     The new customer ID is:
```

и далее значение идентификатора нового покупателя.

§5.4. Упражнение
«Создание хранимых процедур в Management Studio»

Создание хранимой процедуры с использованием конструктора запросов

1. Настроиться на работу с базой данных SQL2005new и убедиться, что в ней есть все пять таблиц (Группы, Должности, Кафедры, Преподаватели, Слушатели).

Установить внутреннее соединение между таблицами Должности и Преподаватели по полю Код_должности (первичный ключ в таблице Должности, внешний – в таблице Преподаватели). Первичный ключ создается через контекстное меню столбца ➢***Изменить***, далее ➢***Создать первичный ключ***; внешний ключ – также через контекстное меню столбца ➢***Изменить***,

далее ➢***Отношения*** ❑***Добавить Спецификация таблиц и столбцов*** [...]

В результате должна появиться конструкция INNER JOIN.

2. Создать в *конструкторе* запросов запрос на выборку всех преподавателей с должностями – через контекстное меню окна редактора запросов ➢***Создать запрос в редакторе…*** (заметим, что при этом запрос создается именно в *конструкторе*, а не в редакторе). Между графическими изображениями таблиц должна быть связь. Далее просто пометить ☑ поля Должность, Фамилия, Имя, Отчество и отследить текст запроса в окне ☐***Конструктор запросов*** (рис. 5)**.** Скопировать текст запроса в буфер обмена.

3. Выполнить запрос через ❑***ОК*** и убедиться, что он работает правильно (7 строк).

4. Создать хранимую процедуру через узел обозревателя объектов:

Программирование → Хранимые процедуры,

далее в контекстном меню узла выполнить ➢***Создать хранимую процедуру…***

5. В шаблон хранимой процедуры вставить оператор SELECT из п. 2.

Изменить шаблон хранимой процедуры:

- в CREATE PROCEDURE изменить имя процедуры на **`[Преподаватель_должность]`**;
- добавить входной целочисленный параметр **`@code int`**;
- оператор SELECT расширить конструкцией

```
WHERE Преподаватели.Код_должности = @code
```

6. Выполнить запрос на создание хранимой процедуры, сформированный в пп. 4–5.

Убедиться, что процедура [Преподаватель_должность] добавлена в список хранимых процедур базы данных SQL2005new.

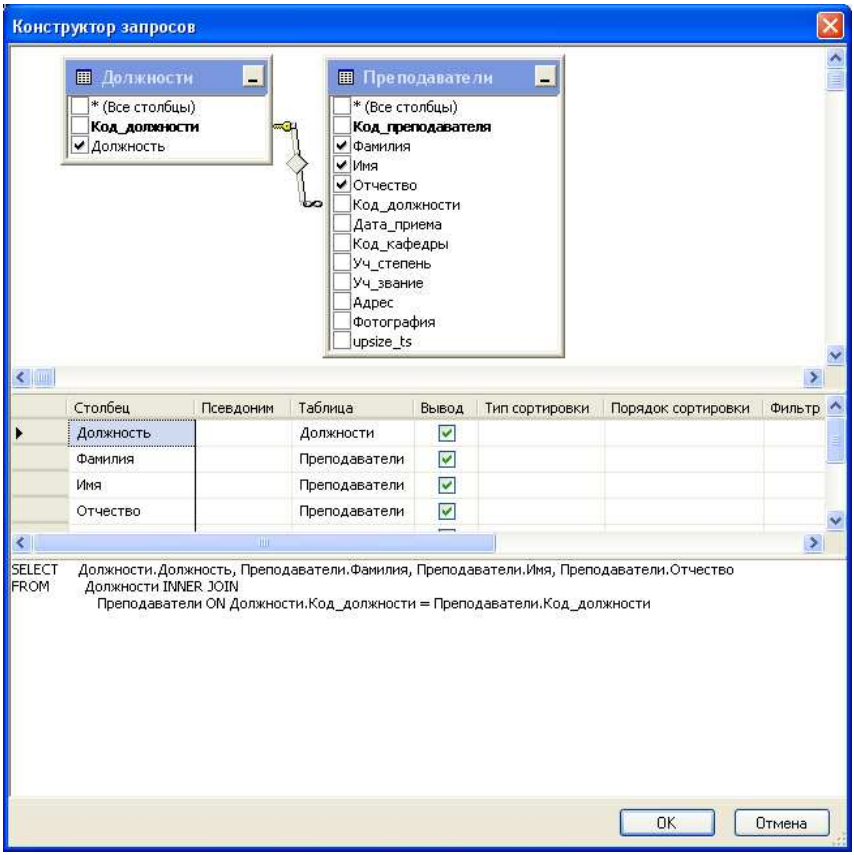


Рис. 5 — Конструирование запроса

7. Выполнить хранимую процедуру с помощью кода T-SQL:

```
EXEC [Преподаватель_должность]
@code = 5
```

В результате информация о всех доцентах (с кодом должности 5) появится во вкладке 🗀***Результаты*** (4 строки).

Замечание. Лучше выполнить хранимую процедуру через ее контекстное меню в обозревателе объектов ➢***Создать сценарий хранимой процедуры*** ➢***Используя EXECUTE*** или ➢***Выполнить хранимую процедуру...***

§ 6. ПРАКТИЧЕСКОЕ ЗАНЯТИЕ 6
Представления

В первом упражнении занятия *«Создание и модификация представления»* (§ 6.1) рассматриваются все операции с представлениями, кроме вызова (создание, модификация, удаление).

Собственно вызов представления с целью просмотра, добавления, модификации, удаления данных описан в упражнении *«Доступ к данным с помощью представления AuthorNames»* (§ 6.2).

§6.1. Упражнение *«Создание и модификация представления»*

Общее замечание. Предполагается, что база данных BookShopDB, над которой строятся представления, должна быть корректной: база должна быть создана, проверена, наполнена и целостна.

Создание представления BookAuthorView в базе данных BookShopDB

1. В окне редактора запросов (Management Studio) ввести и исполнить код T-SQL:

```
USE BookShopDB
GO
CREATE VIEW BookAuthorView
AS
SELECT a.FirstName, a.LastName, b.Title
FROM Authors a JOIN BookAuthors ba
ON a.AuthorID = ba.AuthorID
JOIN Books b
ON ba.TitleID = b.TitleID
```

Заготовка этого и последующих запросов – в файле *Создание и модификация представлений.txt*.

Оператор CREATE VIEW создает представление BookAuthorView в базе данных BookShopDB, которое содержит запрос SELECT, соединяющий таблицу Authors с таблицей BookAuthors, а таблицу BookAuthors – с таблицей Books. В ре-

зультирующий набор этого запроса войдут имена и фамилии авторов, а также названия написанных ими книг.

После исполнения кода T-SQL во вкладке 🗀***Сообщения*** выведется сообщение об успешном выполнении команды.

2. В обозревателе объектов найти созданное представление (через ***Базы данных*** → ***BookShopDB*** → ***Представления***); обратить внимание на вложенные узлы ***Столбцы*** и ***Индексы***.

3. Раскрыть узел ***Столбцы*** и убедиться, что в него входят три столбца, описанные в запросе SELECT из оператора CREATE VIEW.

Модификация представления BookAuthorView из базы данных BookShopDB

4. В окне редактора запросов ввести и исполнить следующий код T-SQL:

```
USE BookShopDB
GO
ALTER VIEW BookAuthorView
AS
SELECT a.FirstName, a.LastName, b.TitleID,
b.Title
FROM Authors a JOIN BookAuthors ba
ON a.AuthorID = ba.AuthorID
JOIN Books b
ON ba.TitleID = b.TitleID
```

Замечание. В принципе, можно скопировать код из п. 1 и внести туда строку ALTER VIEW.

После выполнения кода в представлении должен появиться дополнительный столбец TitleID.

5. Открыть узел ***Столбцы*** представления BookAuthorView и убедиться в появлении столбца TitleID.

Удаление представления BookAuthorView из базы данных BookShopDB

6. В окне редактора запросов ввести и выполнить следующий код T-SQL:

```
USE BookShopDB
GO
DROP VIEW BookAuthorView
```

Этот код удаляет представление BookAuthorView из базы данных BookShopDB**.**

7. В обозревателе объектов найти узел для базы данных ***BookShopDB*** и убедиться в отсутствии представления BookAuthorView.

Замечание. Если представление BookAuthorView все еще находится в списке объектов БД, выполнить команду контекстного меню базы данных ➢***Обновить.***

§6.2. Упражнение «*Доступ к данным с помощью представления AuthorNames*»

Создание представления AuthorNames для просмотра данных

1. В окне редактора запросов ввести код T-SQL:

```
USE BookShopDB
GO
CREATE VIEW AuthorNames
AS
SELECT FirstName, LastName
FROM Authors
```

Этот текст (как и последующие тексты) удобно взять из файла *Доступ к данным через представления.txt*.

Этот оператор создает представление, возвращающее имена и фамилии авторов из таблицы Authors базы данных BookShopDB**.**

2. Исполнить введенный в п. 1 оператор и убедиться в появлении представления AuthorNames (во вкладке 🗀***Сообщения*** должно быть сообщение об успешном выполнении команды).

Просмотр данных через представление AuthorNames

3. В окне редактора запросов ввести код T-SQL:

```
USE BookShopDB
SELECT *
FROM AuthorNames
ORDER BY LastName
```

4. Исполнить код п. 3 и убедиться, что он извлекает данные через представление AuthorNames; результирующий набор

данных упорядочивается по фамилии автора (результат будет в виде таблицы во вкладке 🗀***Результаты***).

Замечание. Если результирующий набор, сформированный в п. 4, окажется пустым (это говорит о том, что таблица Authors базы данных BookShopDB не наполнена), можно выполнить код T-SQL для наполнения этой таблицы, например, из соответствующих полей таблицы Authors базы данных Pubs:

```
USE BookShopDB
INSERT INTO Authors (FirstName, LastName)
SELECT au_fname, au_lname
FROM pubs.dbo.authors
```

После этого повторное выполнение п. 4 даст непустой результирующий набор (23 строки).

Добавление данных через представление AuthorNames

5. В окне редактора запросов ввести следующий код T-SQL:

```
USE BookShopDB
INSERT AuthorNames
VALUES ('William', 'Burroughs')
```

6. Исполнить оператор п. 5, добавляющий сведения о новом авторе через представление AuthorNames**.** Во вкладке 🗀***Сообщения*** выводится сообщение об успешном выполнении команды.

7. Для того, чтобы удостовериться, что строка о новом авторе занеслась *в таблицу Authors*, выполнить в окне редактора запросов код T-SQL:

```
SELECT * FROM Authors
```

Результирующий набор будет во вкладке 🗀***Результаты***; достаточно его «прокрутить» в конец, чтобы увидеть последнюю строку для автора William Burroughs, а также значения «N/A», которые были присвоены по умолчанию полям YearBorn, YearDied и Description.

Модификация данных
с помощью представления AuthorNames

8. В окне редактора запросов ввести следующий код T-SQL:

```
USE BookShopDB
UPDATE AuthorNames
SET FirstName = 'John'
WHERE LastName = 'Burroughs'
```

9. Выполнить код п. 8, изменяющий имя автора с фамилией Burroughs с William на John. Во вкладке 🗀***Сообщения*** должно появиться сообщение об успешном выполнении команды.

10. Чтобы убедиться в модификации данных, следует набрать и выполнить в окне редактора запросов код T-SQL:

```
SELECT * FROM Authors
```

Результирующий набор будет виден во вкладке 🗀***Результаты***; можно по его последней строке убедиться, что имя автора William Burroughs изменено на John Burroughs.

Удаление данных
через представление AuthorNames

11. В окне редактора запросов ввести следующий код T-SQL:

```
USE BookShopDB
DELETE AuthorNames
WHERE LastName = 'Burroughs'
```

12. Выполнить код п. 11, удаляющий автора с фамилией Burroughs из таблицы Authors. Убедиться, что во вкладке 🗀***Сообщения*** появилось сообщение об успешном выполнении команды.

13. Чтобы убедиться в удалении данных, следует набрать и выполнить в окне редактора запросов код T-SQL:

```
SELECT * FROM Authors
```

Результирующий набор будет можно увидеть на вкладке 🗀***Результаты***; нужно его «прокрутить» до конца и убедиться, что автор John Burroughs удален из таблицы Authors.

Замечание. Само представление AuthorNames можно удалить в конце занятия (например, через контекстное меню ➢***Удалить*** в обозревателе объектов Management Studio).

§ 7. ПРАКТИЧЕСКОЕ ЗАНЯТИЕ 7
Триггеры

Занятие начинается иллюстрацией и проверкой основных принципов ссылочной целостности (сначала без использования триггеров вообще) – упражнение *«Применение ограничений каскадной ссылочной целостности»* (§ 7.1).

Полный технологический цикл работы с триггерами (создание, просмотр, проверка, модификация, переименование, отключение, удаление) описан в упражнении *«Создание триггеров и управление ими»* (§ 7.2).

Занятие заканчивается рассмотрением элементов программирования для триггеров (нюансы работы для INSERT, UPDATE, DELETE; псевдотаблицы Inserted и Deleted) – упражнение *«Создание триггера для обновления значения столбца»* (§ 7.3).

§7.1. Упражнение *«Применение ограничений каскадной ссылочной целостности»*

Общее замечание. Предполагается, что база данных BookShopDB, над которой строятся триггеры, должна быть корректной: база должна быть создана, проверена, наполнена и целостна. Соображения по целостности особенно важны для настройки каскадной ссылочной целостности (которая, в частности, может поддерживаться триггерами).

Настройки каскадной ссылочной целостности для ключа TitleID из базы данных BookShopDB

1. В окне редактора запросов (Management Studio) ввести и исполнить код T-SQL:

```
USE BookShopDB
INSERT Orders (CustomerID, EmployeeID, Amount,
OrderDate, DeliveryDate, PaymentID, StatusID)
VALUES (4101, 2101, 30, GetDate(),
DATEADD(day, 5, GetDate()), 11, 41)
INSERT BooksOrders (TitleID, OrderID)
VALUES ('tynm9944', 12101)
```

Заготовка этого и последующих запросов – в файле *Каскадная ссылочная целостность.txt*.

Первый оператор INSERT добавляет заказ в таблицу Orders (обратить внимание, что для OrderDate и DeliveryDate используются функции даты/времени).

Второй оператор INSERT добавляет запись в таблицу BooksOrders; это принципиально важно для проверки каскадной ссылочной целостности на внешний ключ TitleID в таблице BooksOrders.

2. Ввести и исполнить следующий код T-SQL:

```
UPDATE Books
SET TitleID = 'yutr8957'
WHERE TitleID = 'yutr8956'
```

Замечание. Строка с TitleID = yutr8956 *уже должна быть* в таблице Books и в таблице BookAuthors.

Ясно, что такая замена (изменение первичного ключа в таблице Books) *неизбежно* должна привести к нарушению ограничения на значения внешнего ключа (FOREIGN KEY titleid_fk) для таблицы BookAuthors.

Действительно, во вкладке 🗀***Сообщения*** появится примерно такое сообщение об ошибке:

```
Сообщение 547, уровень 16, состояние 0, строка 1
Конфликт инструкции UPDATE с ограничением REFERENCE
"titleid_fk". Конфликт произошел в базе данных
"BookShopDB", таблица "dbo.BookAuthors", столбец
'TitleID'.
Выполнение данной инструкции было прервано.
```

3. Удалить ограничение titleid_fk из таблицы BookAuthors (с целью заменить его более гибкой каскадной ссылочной целостностью):

```
ALTER TABLE BookAuthors
DROP CONSTRAINT titleid_fk
```

4. Добавить каскадную ссылочную целостность к ограничению FOREIGN KEY (titleid_fk) следующим кодом T-SQL:

```
ALTER TABLE BookAuthors
ADD CONSTRAINT titleid_fk FOREIGN KEY (TitleID)
REFERENCES Books (TitleID)
ON UPDATE CASCADE
ON DELETE CASCADE
```

В результате при обновлении первичного ключа TitleID из таблицы Books будет меняться соответствующий внешний ключ в таблице BookAuthors. Таким образом, таблица Books становится *родительской* для таблицы BookAuthors.

5. Выполнить *повторно* код п. 2 и убедиться, что связка Books – BookAuthors работает, но отмечается ошибка, аналогичная п. 2, для связки Books – BooksOrders.

6. Изменить аналогичным образом внешний ключ для таблицы BooksOrders:

```
ALTER TABLE BooksOrders
      DROP CONSTRAINT ordertitleid_fk
ALTER TABLE BooksOrders
      ADD CONSTRAINT ordertitleid_fk
FOREIGN KEY (TitleID) REFERENCES Books (TitleID)
ON UPDATE CASCADE
ON DELETE CASCADE
```

Таким образом, таблица Books становится *родительской* и для таблицы BooksOrders.

7. *Еще раз* выполнить код п. 2 и убедиться в каскадном обновлении ключей в таблицах BookAuthors и BooksOrders (изменение первичного ключа в таблице Books также отследить). Следить можно через обозреватель объектов – выделить имя таблицы и в контекстном меню выполнить ***➢Открыть таблицу.***

А можно выполнить следующий сценарий на T-SQL:

```
SELECT "Books TitleID" = b.TitleID,
       "BookAuthors TitleID" = ba.TitleID,
       "BooksOrders TitleID" = bo.TitleID
FROM Books b INNER JOIN BookAuthors ba
       ON b.TitleID=ba.TitleID
       INNER JOIN BooksOrders bo
       ON b.TitleID=bo.TitleID
WHERE b.TitleID = 'yutr8957'
```

8. Попробовать (но аккуратно) каскадное удаление записей через исполнение кода

```
DELETE FROM Books WHERE TitleID = 'yutr8957'
```

Проверку удаления удобно делать повторным выполнением кода п. 7.

9. Восстановить базу данных в исходное состояние исполнением кода:

```
  ALTER TABLE BookAuthors
      DROP CONSTRAINT titleid_fk
  ALTER TABLE BooksOrders
      DROP CONSTRAINT ordertitleid_fk
  ALTER TABLE BookAuthors
      ADD CONSTRAINT titleid_fk
      FOREIGN KEY (TitleID)
      REFERENCES Books (TitleID)
  ALTER TABLE BooksOrders
      ADD CONSTRAINT ordertitleid_fk
      FOREIGN KEY (TitleID)
      REFERENCES Books (TitleID)
INSERT Books (TitleID, Title, Publisher, PubDate,
      Edition, Cost, SRP, ConditionID, Sold)
VALUES ('yutr8956', 'Sense and Sensibility', 'N/A',
      1/1/1811, 1, 3000, 5900, 5, 0)
      GO
```

§7.2. Упражнение «*Создание триггеров и управление ими*»

Общее замечание. Данное упражнение использует модельный триггер, который просто выдает сообщения о своем срабатывании в разных режимах. Пока действия триггера намеренно упрощены, чтобы прояснить технологию управления триггером. Намного более сложное упражнение на программирование триггеров содержится в § 7.3.

Создание простых триггеров для таблицы Authors базы данных BookShopDB

1. В окне редактора запросов ввести и исполнить код T-SQL:

```
  USE BookShopDB
  GO
  CREATE TRIGGER dbo.insertindicator
  ON dbo.Authors
  AFTER INSERT
  AS
  PRINT 'The insert trigger fired. '
```

Заготовку этого кода (как и последующих) удобно взять из файла *Создание триггеров и управление ими.txt*.

Таким образом, триггер InsertIndicator, «привязанный» к таблице Authors, сработает при любом добавлении данных в таблицу Authors; в итоге во вкладке 🗀***Сообщения*** появится соответствующее сообщение.

2. «Навесить» второй и третий аналогичные триггеры на таблицу Authors:

```
CREATE TRIGGER dbo.updateindicator
ON dbo.Authors
AFTER UPDATE
AS
PRINT 'The update trigger fired.'
GO
CREATE TRIGGER dbo.deleteindicator
ON dbo.Authors
AFTER DELETE
AS
IF @@ROWCOUNT <> 0
PRINT 'The delete trigger fired.'
```

Таким образом, триггеры UpdateIndicator и DeleteIndicator, «привязанные» к той же таблице Authors, выдают соответствующие сообщения, только в случае удаления сообщение появляется тогда и только тогда, когда удалена хотя бы одна запись.

Замечание. Созданные триггеры можно увидеть в обозревателе объектов, «пройдя по цепочке» уровней: ***Базы данных*** → ***BookShopDB*** → ***Таблицы*** → ***dbo.Authors*** → ***Триггеры***.

Проверка триггеров таблицы Authors

3. Ввести и выполнить код T-SQL:

```
INSERT INTO Authors (FirstName, LastName,
YearBorn, YearDied, Description)
VALUES ( 'Max' , 'Doe' , 1962, 'N/A', 'N/A')
```

Убедиться, что занеслась запись в таблицу Authors и одновременно во вкладке 🗀***Сообщения*** появилось сообщение триггера.

4. Ввести и выполнить следующий код T-SQL:

```
UPDATE Authors
SET Authors. FirstName = 'Tucker'
```

```
WHERE Authors.FirstName = 'Max'
```

Убедиться, что запись в таблице Authors изменилась по сравнению с п. 3 и одновременно во вкладке 🗀***Сообщения*** появилось сообщение триггера.

5. Ввести и выполнить следующий код T-SQL:

```
DELETE Authors
WHERE FirstName = 'Tucker'
```

Убедиться, что запись из таблицы Authors удалилась, и сообщение триггера появилось во вкладке 🗀***Сообщения***.

Переименование, модификация и просмотр триггера

6. В окне редактора запросов ввести и исполнить команду запуска хранимой процедуры:

```
sp_rename @objname=insertindicator,
          @newname=insupdcontrol
```

Произойдет переименование триггера из InsertIndicator в InsUpdControl – это делается для того, чтобы триггер стал срабатывать и на INSERT, и на UPDATE.

Замечание. Косвенно о переименовании триггера свидетельствует сообщение на вкладке 🗀***Сообщения*** о том, что переименование может повлечь за собой сбои в работе сценариев и хранимых процедур.

7. Вывести список триггеров, применяемых к таблице Authors:

```
sp_helptrigger @tabname = Authors
```

8. Ввести и выполнить в окне редактора запросов код T-SQL для модификации триггера:

```
ALTER TRIGGER dbo.insupdcontrol
ON dbo.Authors
INSTEAD OF INSERT, UPDATE
AS
PRINT 'Inserts and updates are not allowed at
       this time.'
```

В результате изменится тип триггера (на INSTEAD OF), и тело триггера превратится во временно *запрещающее* все операции добавления и обновления в таблице Authors.

Замечание. Можно включать и выключать триггеры через оператор ALTER TABLE (ключевые слова DISABLE TRIGGER и ENABLE TRIGGER) – см. ниже п. 11.

9. Ввести и выполнить код п. 3. Убедиться, что сработал триггер, установленный в п. 8, и появилось сообщение о том, что обновление в данный момент запрещено. Также убедиться, что в таблицу Authors не добавлено ни одной записи (через SELECT * FROM Authors или пункт контекстного меню ***➢Открыть таблицу***).

10. Просмотреть текст триггера InsUpdControl по хранимой процедуре sp_helptext:

```
sp_helptext @objname = insupdcontrol
```

Отключение и удаление триггера

11. Отключить триггер InsUpdControl следующим оператором T-SQL:

```
ALTER TABLE Authors DISABLE TRIGGER insupdcontrol
```

Убедиться, что на пиктограмме триггера InsUpdControl в обозревателе объектов появилась красная стрелка.

12. Повторно выполнить код п. 3 и убедиться, что запись беспрепятственно добавлена (поскольку триггер отключен).

13. Для удаления триггеров ввести и выполнить в окне редактора запросов код T-SQL:

```
DROP TRIGGER insupdcontrol, updateindicator, deleteindicator
DELETE Authors WHERE FirstName = 'Max'
```

Таким образом, все три созданных ранее триггера удалятся и «лишняя» запись, добавленная в п. 12, тоже.

§7.3. Упражнение «*Создание триггера для обновления значения столбца*»

Общее замечание. В данном упражнении триггер записывает значение «1» в поле Sold таблицы Books базы данных BookShopDB; если заказчик возвращает книгу, то Sold возвращается в исходное состояние – «0» (книга не продана).

Создание триггера AFTER INSERT для таблицы BooksOrders

1. В окне редактора запросов ввести и исполнить код T-SQL:

```
CREATE TRIGGER dbo.update_book_status
```

```
ON dbo.BooksOrders
AFTER INSERT
AS
UPDATE Books
SET Sold = 1
WHERE TitleID =
(SELECT bo.TitleID
FROM BooksOrders bo INNER JOIN inserted i
ON bo.OrderID = i.OrderID)
```

Заготовку этого кода (как и последующих) удобно взять из файла *Программирование триггеров.txt*.

Таким образом, создается триггер Update_Book_Status, «привязанный» к таблице BooksOrders из базы данных BookShopDB. Этот триггер срабатывает при *добавлении* данных к таблице BooksOrders, в результате чего обновляется поле Sold в таблице Books для книги с соответствующим идентификатором (TitleID).

Добавление в триггер обработки удаления и обновления данных таблицы BooksOrders

1. Ввести и исполнить следующий код T-SQL:

```
ALTER TRIGGER dbo.update_book_status
ON dbo.BooksOrders
AFTER INSERT, DELETE
AS
SET NOCOUNT ON
IF EXISTS (SELECT * FROM inserted)
BEGIN
 UPDATE Books
 SET Sold = 1
 WHERE TitleID =
 (SELECT bo.TitleID
 FROM BooksOrders bo INNER JOIN inserted i
 ON bo.OrderID = i.OrderID)
END
ELSE
BEGIN
 UPDATE Books
 SET Sold = 0
 WHERE TitleID =
 (SELECT d.TitleID
```

```
    FROM Books b INNER JOIN deleted d
    ON b.TitleID = d.TitleID)
END
```

Таким образом, триггер Update_Book_Status модифицируется с целью присоединения реакции на событие DELETE. При удалении данных из таблицы BooksOrders этот триггер срабатывает, в результате чего в поле Sold с соответствующим TitleID заносится «0» (что соответствует тому, что книга возвращается в число непроданных). Для бизнес-логики данной БД это принципиально.

Замечание. Когда выполняется ELSE, то оператор SELECT работает уже с идентификатором TitleID *из таблицы Books*, а не из таблицы BooksOrders, так как удаленных записей в BooksOrders уже нет и не с чем сравнивать TitleID в псевдотаблице Deleted.

3. Теперь нужно перейти к *обновлениям*. Дело в том, что единственное обновление в таблице BooksOrders, имеющее отношение к таблице Books, – это обновление столбца TitleID. Даже изменение OrderID не приводит к изменению значения поля Sold в таблице Books (оно должно остаться равным «1»), а вот если изменится TitleID, то нужно установить Sold=0 для «старого» значения TitleID и Sold=1 для «нового» значения (в таблице BooksOrders). В связи с этим триггер опять меняется:

```
ALTER TRIGGER dbo.update_book_status
ON dbo.BooksOrders
AFTER INSERT, UPDATE, DELETE
AS
SET NOCOUNT ON
IF EXISTS (SELECT * FROM inserted)
BEGIN
    UPDATE Books
    SET Sold = 1
    WHERE TitleID =
    (SELECT bo.TitleID
    FROM BooksOrders bo INNER JOIN inserted i
    ON bo.OrderID = i.OrderID)
END
IF EXISTS (SELECT * FROM deleted)
BEGIN
    UPDATE Books
```

```
    SET Sold = 0
    WHERE TitleID =
    (SELECT d.TitleID
    FROM Books b INNER JOIN deleted d
    ON b.TitleID = d.TitleID)
END
```

Исправлений триггера всего два: добавлено событие UPDATE и *убрано ELSE* (всего-то!). Дело в том, что событие UPDATE всегда создает *две* псевдотаблицы – Inserted и Deleted, поэтому в первой части кода полю Sold присваивается «1» для «нового» TitleID, а во второй части кода полю Sold присваивается «0» для «старого» TitleID (которое хранится в псевдотаблице Deleted).

Проверка триггера

4. Добавить новый заказ в таблицу Orders следующим кодом T-SQL:

```
SET IDENTITY_INSERT Orders ON
INSERT INTO Orders
(OrderID, CustomerID, EmployeeID, Amount,
OrderDate, DeliveryDate,PaymentID, StatusID)
VALUES (12108, 4101, 2101, 5, GETDATE(),
GETDATE()+5, 21, 31)
SET IDENTITY_INSERT Orders OFF
GO
```

Убедиться, что появилась строка в таблице Orders.

Замечание. Конструкции SET в начале и в конце кода включают и выключают режим *явной* вставки значений в столбцы идентификаторов (OrderID, CustomerID, EmployeeID) таблицы Orders.

5. Добавить новую запись в таблицу BooksOrders (заказ на книгу с идентификатором mnoi5687 для покупателя с идентификатором заказа 12108 – см. п. 4):

```
INSERT INTO BooksOrders
(OrderID, TitleID)
VALUES (12108, 'mnoi5687')
```

Убедиться, что появилась строка в таблице BooksOrders.

6. Проверить, изменилось ли значение поля Sold для этой книги на «1» (должно измениться):

```
SELECT * FROM Books WHERE TitleID = 'mnoi5687'
```

7. Войти в режим обновления таблицы BooksOrders (замена идентификатора книги в заказе):

```
UPDATE BooksOrders
SET TitleID ='aust1234' WHERE TitleID='mnoi5687'
```

8. Убедиться, что в результате обновления (п. 7) значение поля Sold для «старой» книги (mnoi5687) сброшено в «0», а значение поля Sold для «новой» книги (aust1234) установлено в «1»:

```
SELECT * FROM Books WHERE TitleID = 'mnoi5687'
    OR TitleID = 'aust1234'
```

9. Удалить последнюю запись из таблицы BooksOrders:

```
DELETE BooksOrders WHERE TitleID = 'aust1234'
```

10. Убедиться, что значение поля Sold для идентификатора книги, указанного в п. 9 (то есть для aust1234) переустановлено в «0»:

```
SELECT * FROM Books WHERE TitleID = 'aust1234'
```

11. Удалить триггер из таблицы BooksOrders:

```
DROP TRIGGER dbo.update_book_status
```

Замечание. В принципе, все этапы проверки (пп. 5, 6, 7, 8, 9, 10, 11) можно делать контекстным меню Management Studio (для узлов дерева обозревателя объектов ***Таблицы*** → ***Books***, ***Таблицы*** → ***BooksOrders***, ***Таблицы*** → ***BooksOrders*** →***Триггеры***).

§ 8. ПРАКТИЧЕСКОЕ ЗАНЯТИЕ 8. Проектирование и реализация системы безопасности SQL Server

В ходе занятия необходимо проанализировать контингент потенциальных пользователей базы данных BookShopDB (см. § 1.2). В частности, видимо, нужно выделить трех продавцов книжного магазина (staff01, staff02, staff03) и двух управляющих (точнее, управляющего и помощника управляющего – manager01, manager02). Кроме того, нужен разработчик (DevUser) типа системного администратора.

Такому же анализу нужно подвергнуть и объекты базы данных. Выделяются «особые» таблицы (доступные «не для всех») – Employees, Positions, а также чуть более доступные, но, тем не менее, «особые» (Books, Customers, FormOfPayment, BookCondition, BookAuthors, Authors).

Единственное комплексное упражнение занятия (*«Реализация системы безопасности для БД BookShopDB»*, § 8.1) отслеживает полный цикл проектирования и реализации системы безопасности: аутентификация → авторизация → создание разрешений → установление членства в ролях.

§ 8.1. Упражнение *«Реализация системы безопасности для БД BookShopDB»*

Создание учетных записей Windows для группы и отдельных пользователей

1. Вызвать окно командной строки, например, так: ❑***Пуск*** ➢***Выполнить...*** { далее в окне ❒***Запуск программы*** } ***Открыть:*** := cmd ❑***OK***

2. Выполнить *последовательно* команды операционной системы:

```
net user manager01 /add
net user manager02 /add
net user staff01 /add
net user staff02 /add
net user staff03 /add
```

Замечание. Для ускорения ввода однотипных конструкций можно использовать клавишу [→|].

3. Через ❑***Пуск*** ➢***Настройка*** ➢***Панель управления*** ➢***Учетные записи пользователей*** убедиться, что созданы пять учетных записей для соответствующих пользователей.

4. Попробовать объединить пользователей manager01 и manager02 в рабочую группу managers.

Для этого вызвать консоль ☐ ***Управление компьютером*** через ❑***Пуск*** ➢***Выполнить...*** { далее в окне ☐ ***Запуск программы*** } ***Открыть:*** := mmc ❑***OK.***

Файл для данной консоли (по умолчанию) – *C:\WINDOWS\system32\compmgmt.msc*. Если консоль не установлена, выполнить команду ➢***Консоль*** ➢***Открыть...***

В окне консоли добраться до узла ***Группы*** (через ***Локальные пользователи и группы*** → ***Группы***) и далее выполнить ➢***Действия*** ➢***Создать группу...***

В результате должна быть создана группа managers с членами manager01 и manager02.

Настройка аутентификации группы

5. В окне редактора запросов ввести и выполнить код T-SQL:

```
USE BookShopDB
EXEC sp_grantlogin @loginame='BUILTIN\Пользователи'
EXEC sp_grantlogin @loginame = '<имя компьютера>\
                                              managers'
```

Замечания.

- В последнем операторе EXEC вместо конструкции <имя компьютера> нужно ввести конкретное имя компьютера, видное в первой строке обозревателя объектов.
- Заготовку данного и последующих запросов можно взять из файла *Безопасность для БД BookShopDB.txt*.

Убедиться, что в список пользователей SQL Server добавлена локальная группа Пользователи на соответствующем компьютере и группа managers (через узлы обозревателя объектов ***SQL Server*** → ***Безопасность*** → ***Имена входа***).

6. Подключить разработчика – пользователя с регистрационным именем DevUser и паролем devpass:

```
sp_addlogin @loginame='DevUser', @passwd ='devpass'
```

Обновляя имена входа, убедиться, что пользователь DevUser добавлен.

7. Ознакомиться со всеми учетными именами системы безопасности, имеющими право подключаться к SQL Server:

```
sp_helplogins
```

Обратить внимание на созданные по умолчанию учетные имена BIULTIN\Администраторы и sa (system administrator), причем учетная запись sa является «вечной» – удалить ее из списка аутентификации нельзя.

Настройка авторизации учетных записей в БД BookShopDB

8. Установить для учетных записей соответствие с БД BookShopDB:

```
EXEC      sp_grantdbaccess      @loginame='BUILTIN\
          Пользователи', @name_in_db = 'All Staff'
EXEC sp_grantdbaccess @loginame= '<имя компьютера>\
                                        managers'
EXEC sp_grantdbaccess @loginame = 'DevUser'
EXEC sp_grantdbaccess @loginame ='<имя компьютера>\
                                       manager01'
EXEC sp_grantdbaccess @loginame ='<имя компьютера>\
                                         staff01'
EXEC sp_grantdbaccess @loginame ='<имя компьютера>\
                                         staff02'
```

Убедиться (через узлы обозревателя объектов ***BookShopDB → Безопасность → Пользователи***), что учетные записи «закрепились» за базой данных BookShopDB.

Замечания.

- Для краткости группе с «длинным» именем BIULTIN\Пользователи дан псевдоним All Staff.
- Конструкция <имя компьютера> описана в замечаниях к п. 5.
- Запись manager01 создается отдельно от группы managers для назначения специальных привилегий. По тем же причи-

нам записи staff01 и staff02 выделяются из группы Пользователи.

9. Увидеть список пользователей, авторизованных для базы данных BookShopDB :

```
sp_helpuser
```

Обратить внимание, что в списке указано специальное имя dbo – учетное имя владельца базы данных (db_owner), которое автоматически появляется в каждой БД. С этим именем пройти аутентификацию или авторизацию для доступа к базе данных *нельзя*.

Настройка разрешений для авторизированных учетных имен

10. Поскольку нужно реализовать *иерархию* пользователей в рамках системы безопасности, необходимо начинать настройку с самых «массовых» ролей и разрешений. Таким образом, естественно начать с роли public, к которой автоматически приписаны все авторизированные пользователи. Для ознакомления с этой ролью ввести и выполнить следующий код:

```
sp_helprotect @name=NULL, @username='public'
```

По результату выполнения процедуры видно, что у роли public имеются разрешения SELECT практически для всех объектов БД BookShopDB и разрешения EXECUTE для тех процедур, которые приписаны к этой роли.

11. Установить *явно* для роли public разрешения SELECT для всех пользовательских таблиц БД BookShopDB, кроме Employees и Positions. Для таблиц Orders, BooksOrders и OrderStatus роли public предоставить разрешения INSERT, UPDATE и DELETE:

```
GRANT SELECT ON Authors TO public
GRANT SELECT ON BookAuthors TO public
GRANT SELECT ON BookCondition TO public
GRANT SELECT ON Books TO public
GRANT SELECT ON Customers TO public
GRANT SELECT ON FormOfPayment TO public
GRANT SELECT, INSERT, UPDATE, DELETE
              ON Orders TO public
GRANT SELECT, INSERT, UPDATE, DELETE
              ON BooksOrders TO public
GRANT SELECT, INSERT, UPDATE, DELETE
               ON OrderStatus TO public
```

Проверка выполнения – через повторный запуск процедуры sp_helprotect (см. п. 10). Убедиться, что разрешения для пользовательских таблиц переустановлены.

Можно также увидеть разрешения через контекстное меню соответствующей таблицы ➢***Свойства***, вкладка 🗀***Разрешения.***

12. Окончательно сформировать иерархию безопасности:

```
GRANT INSERT, UPDATE, DELETE
ON Authors TO [<имя компьютера>\managers]
GRANT INSERT, UPDATE, DELETE
ON BookAuthors TO [<имя компьютера>\managers]
GRANT INSERT, UPDATE, DELETE
ON BookCondition TO [<имя компьютера>\managers]
GRANT INSERT, UPDATE, DELETE
ON Books TO [<имя компьютера>\managers]
GRANT INSERT, UPDATE, DELETE
ON Customers TO [<имя компьютера>\managers]
GRANT INSERT, UPDATE, DELETE
ON FormOfPayment TO [<имя компьютера>\managers]
GRANT ALL ON Employees TO [<имя компьютера>
                           \managers]
GRANT ALL ON Positions to [<имя компьютера>
                           \managers]
```

Замечание. На последние два оператора GRANT ALL могут быть получены сообщения, что ключевое слово ALL является устаревшим; тем не менее, проверка показывает, что все основные разрешения установлены.

Таким образом, к разрешениям роли public *добавлены* разрешения, которые даны группе managers, являющейся *членом* роли public. Последние два оператора GRANT дают *все* разрешения на работу с таблицами Employees и Positions для группы managers. Между тем, роль public вообще *не имеет* разрешений на работу с этими таблицами.

Назначение членства в фиксированной роли

13. Назначить двух членов в фиксированную роль db_backupoperator для БД BookShopDB:

```
EXEC sp_addrolemember @rolename =
      'db_backupoperator',
      @membername = '<имя компьютера>\staff01'
EXEC sp_addrolemember @rolename =
       'db_backupoperator',
      @membername = '<имя компьютера>\staff02'
```

Таким образом, эти два члена персонала в силу своего членства в роли db_backupoperator могут создавать резервные копии БД (проверка через цепочку узлов обозревателя объектов: ***BookShopDB → Безопасность → Роли → Роли базы данных***).

14. Назначить всем менеджерам право выполнять администрирование системы безопасности SQL Server, а конкретным пользователям manager01 и DevUser – членство в фиксированной роли sysadmin на уровне *сервера* (в результате чего они получают полный доступ к SQL Server):

```
EXEC sp_addsrvrolemember @loginame = '<имя
компьютера>\managers', @rolename = 'securityadmin'
EXEC sp_addsrvrolemember @loginame = '<имя
компьютера>\manager01',@rolename = 'sysadmin'
EXEC sp_addsrvrolemember @loginame =
'DevUser',@rolename = 'sysadmin'
```

Замечание. Параметры процедур пп. 13 и 14 – разные.

Проверка через цепочку узлов обозревателя объектов: ***SQL Server → Безопасность → Серверные роли.***

§ 9. ПРАКТИЧЕСКОЕ ЗАНЯТИЕ 9. Бизнес-аналитика в SQL Server

В ходе занятия выстраивается практически полная «технологическая цепочка» для выполнения многомерной аналитической обработки данных (OLAP – On-Line Analytical Processing):

- создание и заполнение хранилища данных на базе реляционной БД (упражнение *«Создание и заполнение хранилища данных»*, § 9.1);
- создание OLAP-куба на базе хранилища данных (упражнение *«Создание многомерной базы данных»*, § 9.2);
- разворачивание и обработка OLAP-куба (по измерениям и мерам) и дальнейшая работа с ним по двум направлениям: 1) через использование служб анализа (SSAS – SQL Server Analysis Services) – вплоть до бизнес-аналитики и интеллектуального анализа данных; 2) через сводные таблицы Microsoft Excel, использующегося в качестве клиента (упражнение *«Работа с многомерным хранилищем данных»*, § 9.3).

Предполагается, что развитие данной тематики будет осуществляться в рамках следующего курса, входящего в общий профессиональный цикл, – «Эксплуатация информационных систем и баз данных».

§ 9.1. Упражнение *«Создание и заполнение хранилища данных»*

Постановка задачи. На основе БД Northwind создать хранилище данных (ХД) Northwind_Mart и наполнить его данными из БД Northwind и некоторыми преобразованными данными.

Создание хранилища данных Northwind_Mart

1. Построить диаграмму БД Northwind (через контекстное меню ➢***Построить диаграмму базы данных*** узла обозревателя объектов ***Northwind*** → ***Диаграммы базы данных***).

Ознакомиться по диаграмме со связями таблиц (диаграмму можно сохранить).

2. Создать базу данных Northwind_Mart в виде ХД из одной таблицы фактов (Sales_Fact) и пяти таблиц измерений (Time_Dim, Shipper_Dim, Product_Dim, Employee_Dim, Customer_Dim) – через выполнения следующего сценария (полный текст содержится в файле *OLAP.txt*):

```
CREATE DATABASE Northwind_Mart ON PRIMARY
(
NAME=Northwind_Mart_Data,
FILENAME='C:\Program Files\Microsoft SQL Server\
MSSQL.1\MSSQL\Data\Northwind_Mart_Data.MDF',
SIZE=5MB,
FILEGROWTH=10%
)
LOG ON
(
NAME=Northwind_Mart_Log,
FILENAME='C:\Program Files\Microsoft SQL Server\
MSSQL.1\MSSQL\Data\Northwind_Mart_Log.LDF',
SIZE=2MB,
FILEGROWTH=10%
)
GO
USE Northwind_Mart
GO
CREATE TABLE [dbo].[Customer_Dim] (
[CustomerKey] [int] IDENTITY (1, 1) NOT NULL ,
[CustomerID] [nchar] (5) NOT NULL ,
[CompanyName] [nvarchar] (40) NOT NULL ,
[ContactName] [nvarchar] (30) NOT NULL ,
[ContactTitle] [nvarchar] (30) NOT NULL ,
[Address] [nvarchar] (60) NOT NULL ,
[City] [nvarchar] (15) NOT NULL ,
[Region] [nvarchar] (15) ,
[PostalCode] [nvarchar] (10) NULL ,
[Country] [nvarchar] (15) NOT NULL ,
[Phone] [nvarchar] (24) NOT NULL ,
[Fax] [nvarchar] (24) NULL
) ON [PRIMARY]
GO
CREATE TABLE [dbo].[Employee_Dim] (
[EmployeeKey] [int] IDENTITY (1, 1) NOT NULL ,
```

```
[EmployeeID] [int] NOT NULL ,
[EmployeeName] [nvarchar] (30) NOT NULL ,
[HireDate] [datetime] NULL
) ON [PRIMARY]
GO
CREATE TABLE [dbo].[Product_Dim] (
[ProductKey] [int] IDENTITY (1, 1) NOT NULL ,
[ProductID] [int] NOT NULL ,
[ProductName] [nvarchar] (40) NOT NULL ,
[SupplierName] [nvarchar] (40) NOT NULL ,
[CategoryName] [nvarchar] (15) NOT NULL ,
[ListUnitPrice] [money] NOT NULL
) ON [PRIMARY]
GO
CREATE TABLE [dbo].[Sales_Fact] (
[TimeKey] [int] NOT NULL ,
[CustomerKey] [int] NOT NULL ,
[ShipperKey] [int] NOT NULL ,
[ProductKey] [int] NOT NULL ,
[EmployeeKey] [int] NOT NULL ,
[RequiredDate] [datetime] NOT NULL ,
[LineItemFreight] [money] NOT NULL ,
[LineItemTotal] [money] NOT NULL ,
[LineItemQuantity] [smallint] NOT NULL ,
[LineItemDiscount] [money] NOT NULL
) ON [PRIMARY]
GO
CREATE TABLE [dbo].[Shipper_Dim] (
[ShipperKey] [int] IDENTITY (1, 1) NOT NULL ,
[ShipperID] [int] NOT NULL ,
[ShipperName] [nvarchar] (40) NOT NULL
) ON [PRIMARY]
GO
CREATE TABLE [dbo].[Time_Dim] (
[TimeKey] [int] IDENTITY (1, 1) NOT NULL ,
[TheDate] [datetime] NOT NULL ,
[DayOfWeek] [nvarchar] (20) NOT NULL ,
[Month] [int] NOT NULL ,
[Year] [int] NOT NULL ,
[Quarter] [int] NOT NULL ,
[DayOfYear] [int] NOT NULL ,
[Holiday] [nvarchar] (1) NOT NULL ,
```

```
     [Weekend] [nvarchar] (1) NOT NULL ,
     [YearMonth] [nvarchar] (20) NOT NULL ,
     [WeekOfYear] [int] NOT NULL
     ) ON [PRIMARY]
     GO
ALTER TABLE [dbo].[Customer_Dim] WITH NOCHECK
ADD CONSTRAINT [PK_Customer_Dim] PRIMARY KEY
NONCLUSTERED ([CustomerKey]) ON [PRIMARY]
     GO
ALTER TABLE [dbo].[Employee_Dim] WITH NOCHECK
ADD CONSTRAINT [PK_Employee_Dim] PRIMARY KEY
NONCLUSTERED ([EmployeeKey]) ON [PRIMARY]
     GO
ALTER TABLE [dbo].[Product_Dim] WITH NOCHECK
ADD CONSTRAINT [PK_Product_Dim] PRIMARY KEY
NONCLUSTERED ([ProductKey]) ON [PRIMARY]
     GO
ALTER TABLE [dbo].[Sales_Fact] WITH NOCHECK
ADD CONSTRAINT [PK_Sales_Fact] PRIMARY KEY
NONCLUSTERED
([TimeKey],
[CustomerKey],
[ShipperKey],
[ProductKey],
[EmployeeKey]
) ON [PRIMARY]
     GO
ALTER TABLE [dbo].[Shipper_Dim] WITH NOCHECK
ADD CONSTRAINT [PK_Shipper_Dim] PRIMARY KEY
NONCLUSTERED ([ShipperKey]) ON [PRIMARY]
     GO
ALTER TABLE [dbo].[Time_Dim] WITH NOCHECK
ADD CONSTRAINT [PK_Time_Dim] PRIMARY KEY
NONCLUSTERED ([TimeKey]) ON [PRIMARY]
     GO
ALTER TABLE [dbo].[Sales_Fact]
ADD CONSTRAINT [FK_Sales_Fact_Customer_Dim]
FOREIGN KEY ([CustomerKey])
REFERENCES [dbo].[Customer_Dim] ([CustomerKey]),
CONSTRAINT [FK_Sales_Fact_Employee_Dim]
FOREIGN KEY ([EmployeeKey])
REFERENCES [dbo].[Employee_Dim] ([EmployeeKey]),
```

```
CONSTRAINT [FK_Sales_Fact_Product_Dim]
FOREIGN KEY ([ProductKey])
REFERENCES [dbo].[Product_Dim] ([ProductKey]),
CONSTRAINT [FK_Sales_Fact_Shipper_Dim]
FOREIGN KEY ([ShipperKey])
REFERENCES [dbo].[Shipper_Dim] ([ShipperKey]),
CONSTRAINT [FK_Sales_Fact_Time_Dim]
FOREIGN KEY ([TimeKey])
REFERENCES [dbo].[Time_Dim] ([TimeKey])
```

3. Построить диаграмму ХД Northwind_Mart (рис. 6) и убедиться, что схема «звезда» поддерживается отношениями «один к многим» (диаграмму можно сохранить).

Формирование таблицы Customer_Dim

4. Поскольку столбцы таблицы Customer_Dim совпадают со столбцами таблицы Customers БД Northwind, можно воспользоваться мастером импорта / экспорта через контекстное меню ХД Northwind_Mart ➢***Задачи*** ➢***Импорт данных.*** В качестве источника импорта выбрать OLE DB Provider for SQL Server, БД Northwind; в качестве назначения – достаточно SQL Native Client (в принципе, можно взять опять OLE DB Provider for SQL Server), БД Northwind_Mart. В ходе работы мастера можно выполнять предварительный просмотр таблицы, вносить изменения и т.п.; в результате в таблицу Customer_Dim будет передана 91 строка по *одинаковым* именам столбцов.

Замечание. В ходе работы мастера импорта / экспорта формируется и выполняется пакет для служб интеграции SSIS (SQL Server Integration Services); в принципе, можно таблицу Customer_Dim (да и все прочие таблицы) формировать средствами SSIS.

5. Для отказа от пустых значений в таблице Customer_Dim в окне редактора запросов ввести и выполнить код T-SQL:

```
UPDATE Customer_Dim
SET Region = 'Other' WHERE Region IS NULL
```

Убедиться, сравнив две таблицы, что передача данных прошла успешно, при этом в таблицу Customer_Dim добавился суррогатный ключ.

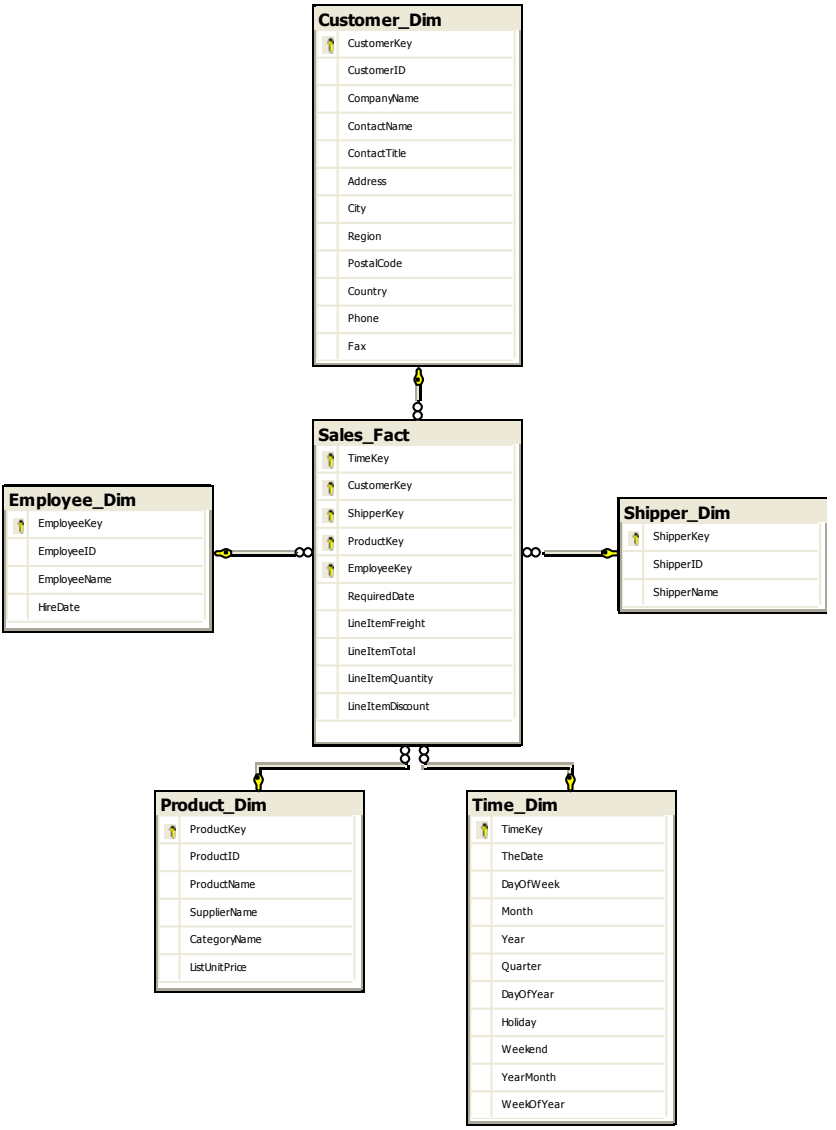


Рис. 6 — Диаграмма ХД Northwind_Mart

Наполнение таблицы Time_Dim

6. В таблицу Time_Dim вносятся многочисленные *меры* (уровни иерархии): дата, день недели, месяц, год, квартал, день года, месяц года, неделя года, а также битовые поля (Y / N) «праздник» и «конец (начало) недели». Для этого нужно ввести и выполнить следующий код:

```
INSERT Time_Dim
(TheDate,DayOfWeek,[Month],[Year],[Quarter],
DayOfYear,Holiday,Weekend, YearMonth,WeekOfYear)
SELECT DISTINCT
S.ShippedDate AS TheDate,
DateName(dw, S.ShippedDate) AS DayOfWeek,
DatePart(mm, S.ShippedDate) AS [Month],
DatePart(yy, S.ShippedDate) AS [Year],
DatePart(qq, S.ShippedDate) AS [Quarter],
DatePart(dy, S.ShippedDate) AS DayOfYear,
'N' AS Holiday,
case DatePart(dw, S.ShippedDate)
when (1) then 'Y'
when (7) then 'Y'
else 'N'
end
AS Weekend,
DateName(month, S.ShippedDate) +
'_' + DateName(year,S.ShippedDate) AS YearMonth,
DatePart(wk, S.ShippedDate) AS WeekOfYear
FROM Northwind.dbo.Orders S
WHERE S.ShippedDate IS NOT NULL
```

В результате в таблицу Time_Dim добавится 387 строк.

Наполнение таблицы Product_Dim

7. Наполнение идет с помощью внутреннего соединения таблиц Products, Categories и Suppliers; для наполнения следует ввести и выполнить следующий код:

```
INSERT Product_Dim
(ProductID, ProductName, SupplierName,
CategoryName, ListUnitPrice)
SELECT p.ProductID, p.ProductName,
s.CompanyName AS SupplierName, c.CategoryName,
p.UnitPrice AS ListUnitPrice
```

```
FROM Northwind.dbo.Categories  c
     INNER JOIN Northwind.dbo.Products p
     ON c.CategoryID = p.CategoryID
     INNER JOIN Northwind.dbo.Suppliers s
     ON s.SupplierID = p.SupplierID
```

В результате в таблицу Product_Dim будет добавлено 77 строк.

Наполнение таблицы Employee_Dim

8. Здесь достаточно *одной* таблицы Employee из БД Northwind, но предлагается объединить в поле EmployeeName имя и фамилию служащего:

```
INSERT Employee_Dim (EmployeeID, EmployeeName,
       HireDate)
SELECT  e.EmployeeID, e.FirstName + ' ' +
        e.LastName AS EmployeeName, e.HireDate
        FROM Northwind.dbo.Employees e
```

В результате в таблицу Employee_Dim будет добавлено 9 строк.

Наполнение таблицы Shipper_Dim

9. Здесь единственная тонкость – нужно брать в качестве поля ShipperName значение поля CompanyName из таблицы Shippers БД Northwind:

```
INSERT Shipper_Dim (ShipperID, ShipperName)
SELECT  s.ShipperID AS ShipperID,
        s.CompanyName AS ShipperName
FROM Northwind.dbo.Shippers s
```

В результате в таблицу Shipper_Dim будет добавлено 3 строки.

Наполнение таблицы фактов

10. Это самая большая и относительно стабильная таблица (протокол произведенных сделок), имеющая шесть измерений (включая дату) и три агрегатных данных:

```
INSERT Sales_Fact (TimeKey, CustomerKey,
       ShipperKey, ProductKey, EmployeeKey,
```

```
            RequiredDate, LineItemFreight,
            LineItemTotal, LineItemQuantity,
            LineItemDiscount)
     SELECT
Northwind_Mart.dbo.Time_Dim.TimeKey AS TimeKey,
Northwind_Mart.dbo.Customer_Dim.CustomerKey
          AS CustomerKey,
Northwind_Mart.dbo.Shipper_Dim.ShipperKey
          AS ShipperKey,
Northwind_Mart.dbo.Product_Dim.ProductKey
          AS ProductKey,
Northwind_Mart.dbo.Employee_Dim.EmployeeKey
          AS EmployeeKey,
Northwind.dbo.Orders.RequiredDate AS RequiredDate,
Orders.Freight * Northwind.dbo.[Order Details].
          Quantity / (SELECT SUM(Quantity)
          FROM Northwind.dbo.[Order Details] od
          WHERE od.OrderID = Orders.OrderID)
          AS LineItemFreight,
[Order Details].UnitPrice *[Order Details].Quantity
          AS LineItemTotal,
[Order Details].Quantity AS LineItemQuantity,
[Order Details].Discount*[Order Details].UnitPrice*
          [Order Details].Quantity
          AS LineItemDiscount
FROM Northwind.dbo.Orders
     INNER JOIN Northwind.dbo.[Order Details]
      ON Orders.OrderID = [Order Details].OrderID
     INNER JOIN Northwind_Mart.dbo.Product_Dim
      ON [Order Details].ProductID =
      Northwind_Mart.dbo.Product_Dim.ProductID
     INNER JOIN Northwind_Mart.dbo.Customer_Dim
      ON Orders.CustomerID =
      Northwind_Mart.dbo.Customer_Dim.CustomerID
     INNER JOIN Northwind_Mart.dbo.Time_Dim
      ON Orders.ShippedDate =
                Northwind_Mart.dbo.Time_Dim.TheDate
     INNER JOIN Northwind_Mart.dbo.Shipper_Dim
      ON Orders.ShipVia =
           Northwind_Mart.dbo.Shipper_Dim.ShipperID
     INNER JOIN Northwind_Mart.dbo.Employee_Dim
      ON Orders.EmployeeID =
```

```
        Northwind_Mart.dbo.Employee_Dim.EmployeeID
    WHERE (Orders.ShippedDate IS NOT NULL)
```

Таблица Sales_Fact получается *очень* большой (2082 строки), поскольку выполняется внутреннее соединение семи (!) таблиц.

§ 9.2. Упражнение *«Создание многомерной базы данных»*

1. Запустить Business Intelligence Development Studio (BIDS) и выбрать пакет «Службы SSAS (Analysis Services)» (потребуется дополнительное соединение с SQL Server).

Удобнее это делать через создание *проекта*:

➢***Файл*** ➢***Создать*** ➢***Проект*** ⇩ ***Проект служб SSAS***

и в появившемся диалоговом окне дать проекту название, например, Northwind_OLAP.

Описание источников данных для OLAP-кубов

Общее замечание. В этом и последующих пунктах сценариев предполагается, что работа выполняется в диалоге с соответствующим *мастером*. Диалоговые окна мастеров достаточно информативны и интуитивно понятны. В связи с этим полные диалоги с мастерами здесь не описываются, а лишь акцентируются некоторые особенности конкретного мастера и обязательные параметры, которые должны быть установлены в соответствующем диалоговом окне.

2. В обозревателе решений через контекстное меню ➢***Создать источник данных*** запустить соответствующий мастер. В диалоговом окне нажать ❑***Создать*** и выбрать имя сервера и имя базы данных (Northwind_Mart). Можно проверить соединение с сервером по ❑***Проверить соединение.*** Далее для учетных данных пользователя установить ⊙***По умолчанию***, просмотреть строку соединения и нажать ❑***Готово.***

В итоге образуется файл *Northwind Mart.ds* (обратить внимание, что внутреннее подчеркивание в имени файла снялось; *ds* означает data source).

Создание представления источника данных

3. Нужно сформировать представление типа «звезда», чтобы было четко видно, где измерения, а где агрегаты. Для этого в контекстном меню обозревателя решений выполнить пункт ➢***Создать представление источника***, в результате чего стартует соответствующий мастер, *уже настроенный* на реляционный источник данных Northwind Mart. На следующем шаге работы мастера нужно включить все объекты источника (таблицы), *кроме dbo.sysdiagrams*.

По окончании нужно открыть представление и убедиться, что создана схема типа «звезда» (рис. 7). Можно также по контекстному меню *таблицы* ➢***Просмотр данных*** вызвать диалоговое окно с вкладками 🗀***Таблица***, 🗀***Сводная таблица***, 🗀***Диаграмма***, 🗀***Сводная диаграмма.*** Таким образом, можно работать со сводной таблицей и сводной диаграммой там, где есть интегрирующие поля (в нашем случае такими полями обладает, например, таблица Product_Dim).

Наконец, можно увидеть XML-код представления через контекстное меню *представления*, пункт ➢***Перейти к коду.*** Представление будет иметь имя *Northwind Mart.dsv* (расширение *.dsv* расшифровывается как *data source view*).

Создание OLAP-куба

Замечание. Процесс создания OLAP-куба очень «хитрый» и ответственный; куб с большим количеством измерений, даже если и будет нормально создан, еще неизвестно, как будет обрабатываться. В связи с этим в данном сценарии описывается процесс создания куба с *тремя* измерениями (типа сводной таблицы Microsoft Excel или результата перекрестного запроса Microsoft Access). Напомним, что максимальное количество измерений в БД Northwind_Mart равно пяти.

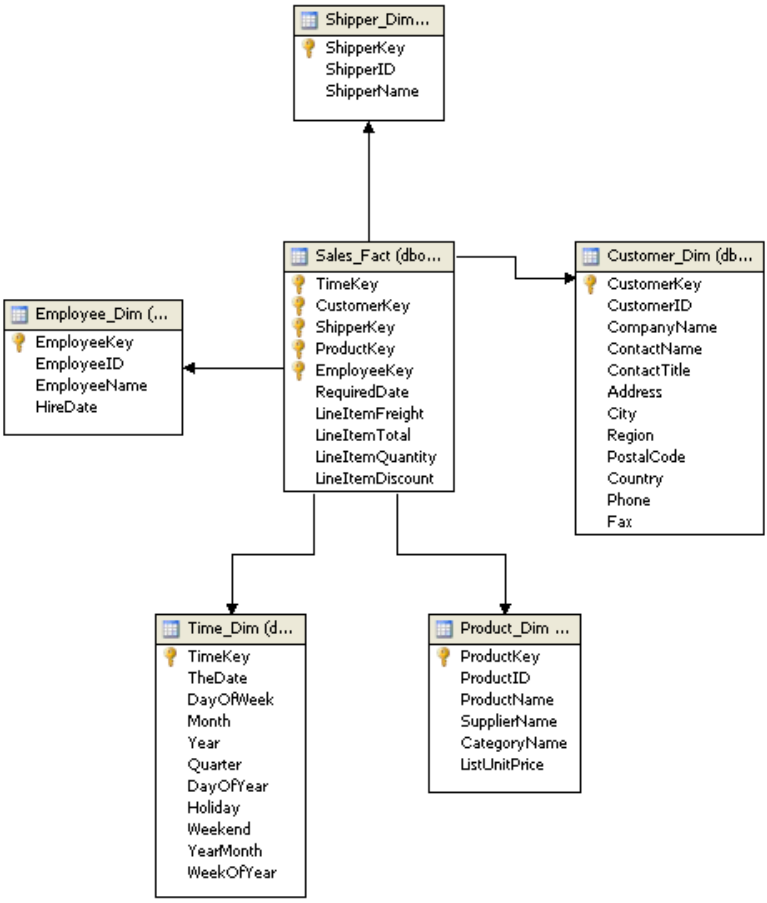


Рис. 7 — Схема типа «звезда»

4. В контекстном меню узла ***Кубы*** обозревателя решений выполнить ➢***Создать куб*** и дождаться запуска соответствующего мастера.

В мастере кубов установить:

⊙***Построить куб с использованием источника данных***

☑***Автоматическое построение*** **(снять)**

Далее выбрать Sales_Fact в качестве таблицы *фактов* и, например, Time_Dim, Product_Dim и Customer_Dim в качестве

таблиц *измерений*. Для таблицы Time_Dim создать *иерархию* вида Year – Quarter – Month – The Date с помощью ***Таблица измерения времени: ⇩ Time_Dim.***

В качестве *мер* можно включить все агрегатные данные (LineItemFreight, LineItemTotal, LineItemQuantity, LineItemDiscount).

Созданному кубу дать имя Northwind Mart (расширение имени файла *.cube*). Готовый куб должен отобразиться в окне представления (рис. 8).

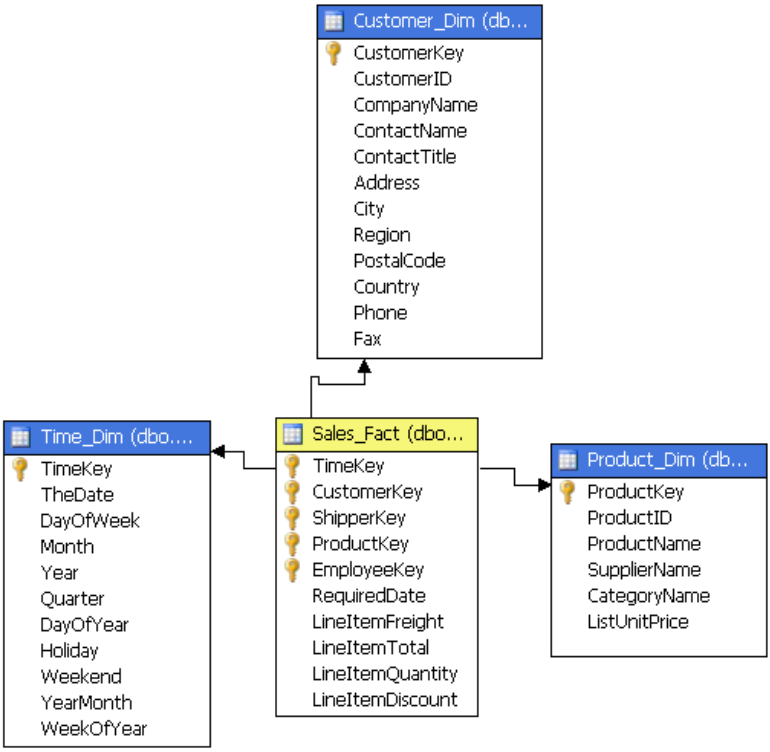


Рис. 8 — Графическое представление куба

§ 9.3. Упражнение
«Работа с многомерным хранилищем данных»

Использование SSAS для обработки OLAP-куба

1. Убедиться, что службы анализа (SSAS) SQL Server запущены и работают (для этого самое удобное – запустить диспетчер конфигурации (Configuration Manager)).

2. Обработать OLAP-куб через пункт его контекстного меню ➢***Обработка….*** В ответ на предложение развернуть проект и далее выполнить обработку необходимо ответить ❑***Да*** (при этом могут появиться сообщения о том, что проект создан в устаревшей версии SQL Server).

Обработка должна завершиться «штатно», после чего следует закрыть окна обработки по ❑***Закрыть.***

3. Запустить обозреватель куба с помощью его контекстного меню ➢***Обзор*** 🗀***Обозреватель.***

Сформировать перетаскиванием мышью (как элементы сводной таблицы) шаблоны поля столбцов (например, поле CompanyName из таблицы Customer_Dim), поля строк (например, поле CategoryName из таблицы Product_Dim), поля фильтра (например, поле Quarter из таблицы Time_Dim), поля данных (*мер*) (например, поле LineItemTotal из таблицы Sales_Fact).

Попробовать осуществить «вращение» куба, то есть менять порядок шаблонов полученной трехмерной таблицы с помощью мыши (это напоминает транспонирование сводной таблицы в Microsoft Excel).

4. Удалить поле Quarter из поля фильтра с помощью контекстного меню ➢***Удалить поле.*** Вместо этого поля перетащить всю автоматически созданную иерархию Year – Quarter – Month – The Date (см. п. 4 § 9.1).

Сворачивать и разворачивать уровни иерархии с помощью кнопок ⊟ и ⊞ .

5. Поработать с *мерами* (агрегатными данными): перетащить второй агрегат в шаблон поля данных (например, поле LineItemDiscount из таблицы Sales_Fact); установить другое (например, процентное) представление данных с помощью пункта контекстного меню ➢***Показать как.***

6. В режиме ознакомления посмотреть основные направления бизнес-аналитики по ➢***Куб*** ➢***Добавить бизнес-аналитику.*** В соответствующем мастере выбрать, например, пункт ***Определить логику операций со временем*** и установить:

☑***Годовой прирост в %***

☑***Квартальный прирост в %***

☑***Месячный прирост в %***

В результате должен сформироваться *сценарий* вычислений, который можно выполнить вкладкой 🗀***Вычисления.***

7. Также в режиме ознакомления запустить мастер *интеллектуального анализа данных* по пункту контекстного меню обозревателя решений ➢***Создать структуру интеллектуального анализа данных.*** В мастере ознакомиться с девятью методами анализа и попробовать запустить, например, метод «Простой Байес».

Можно провести обработку до конца и просмотреть результаты в виде диаграмм и профилей (без интерпретации).

Использование Microsoft Excel как клиента обработки OLAP-куба

8. Запустить Microsoft Excel (версия, в принципе, не принципиальна; в дальнейшем предполагается Microsoft Excel 2010).

9. В меню ➢***Данные*** выполнить пункт ленты 🗀***Из других источников*** ▾***Из служб аналитики.*** В результате запустится мастер подключения к серверу БД, в котором надо выбрать имя базы данных Northwind_Mart. В ответ должен появиться идентификатор *куба* и образоваться файл *Northwind_Mart.odc* для подключения к источнику данных.

10. По окончании работы мастера вызовется диалоговое окно Microsoft Excel, в котором требуется указать режим создания сводной диаграммы и *отчета* сводной таблицы на *новом* листе.

Повторить примерно такие же действия, какие были описаны в пп. 3 и 4 (*по отчету*).

Убедиться, что *все* операции (транспонирование, фильтрация, вычисление процентов и т.п.) работают в полном объеме и параллельно строится соответствующая сводная диаграмма примерно с теми же возможностями выполнения операций (кроме транспонирования).

СПИСОК ЛИТЕРАТУРЫ И ИНТЕРНЕТ-ИСТОЧНИКОВ

1. *Вишневский, А.* Microsoft SQL Server. Эффективная работа / А. Вишневский. – СПб.: Питер, 2009. – 544 с.

2. *Волоха, А.* Microsoft SQL Server 2005 [Электронный ресурс] : новые возможности / А. Волоха. – СПб.: Питер, 2006. – 304 с. – URL: http://www.twirpx.com/file/949076/ (дата обращения 16.01.2017).

3. *Вьейра, Р.* Программирование баз данных Microsoft SQL Server 2005 [Электронный ресурс] : базовый курс / Р. Вьейра. – М.: И.Д. Вильямс, 2008. – 832 с. – (Программистам от программистов). – URL: http://mexalib.com/view/16702 (дата обращения 16.01.2017).

4. *Михеев, Р.* MS SQL Server 2005 для администраторов [Электронный ресурс] : специальный курс / Р. Михеев. – СПб.: БХВ-Петербург, 2007. – 544 с. – URL: http://adm-lib.ru/sisadminu/ms-sql-server-2005-dlya-administratorov.html (дата обращения 16.01.2017).

5. *Морган, С.* Проектирование и оптимизация доступа к базам данных Microsoft SQL Server 2005 : учебный курс Microsoft / С. Морган, Т. Тернстерн. – М.: Русская редакция, 2008. – 480 с.

6. *Нильсен, П.* Microsoft SQL Server 2005 [Электронный ресурс]: библия пользователя / П. Нильсен. – М.: И.Д. Вильямс, 2008. – 1232 с. – URL: http://mexalib.com/author/%D0%9D%D0%B8%D0%BB%D1%8C%D1%81%D0%B5%D0%BD%20%D0%9F (дата обращения 11.01.2017).

7. *Новиков Ф.А., Яценко А.Д.* Microsoft Office 2000 в целом. - СПб.: BHV-Санкт-Петербург, 2000. – 728 с.

8. *Перевозчиков, В.Я.* Разработка и сопровождение баз данных в MS SQL Server 2000 [Электронный ресурс] / В. Я. Перевозчиков. – М.: Лаборатория книги, 2012. – 241 с. – URL: http://biblioclub.ru/index.php?page=book&id=142004&sr=1 (дата обращения 16.01.2017)

9. *Попова, В.Г.* Разработка баз данных СУБД ACCESS: Компьютерный практикум. Часть III / В.Г. Попова, С.Б. Тарасов. – М.: Издательство РАГС, 2006. – 88 с.

10. *Федоров, А.* Введение в OLAP [Электронный ресурс] / А. Федоров, Н. Елманова. – М.: Диалог-МИФИ, 2002. – 60 с. – URL: http://booksteka.info/cat1/kniga1185-down.html (дата обращения 11.01.2017).

11. *Харинатх, С.* SQL Server 2005 Analysis Services и MDX для профессионалов / С. Харинатх, С. Куинн. – М.; СПб.; Киев : Диалектика, 2008. – 848 с.

12. Хранилища данных. Лекция 3. Создание куба в SQL Server 2005 [Электронный ресурс] : презентация. – М.: Национальный Открытый Университет «ИНТУИТ», 2014. – 79 с. – URL: http://biblioclub.ru/index.php?page=book&id=237113 &sr=1 (дата обращения 16.01.2017).

13. Хранилища данных. Лекция 4. Создание многомерного хранилища данных на основе MS SQL Server 2005[Электронный ресурс] : презентация. – М.: Национальный Открытый Университет «ИНТУИТ», 2014. – 67 с. – URL: http://biblioclub.ru/index.php? page=book& id=237114&sr=1 (дата обращения 16.01.2017).

14. *Microsoft Corporation*. Проектирование и реализация баз данных Microsoft SQL Server 2000 [Электронный ресурс] : учебный курс MCAD/MCSE, MCDBA. – Москва : Русская редакция, 2003. – 512 с. – URL: http://www.twirpx.com/file/13575/ (дата обращения 11.01.2017).

15. *https://msdn.microsoft.com/ru-ru/library/ms345332.aspx* – библиотека MSDN уроков фирмы Microsoft по SQL Server.

16. *https://itvdn.com/ru/video/sql-essential* – видеокурс SQL Essential (9 уроков, автор Давид Бояров).

17. *http://playithub.com/watch/qndDTydhD2w/-ms-sql-server-.html* – видеоуроки SQL Server для начинающих.

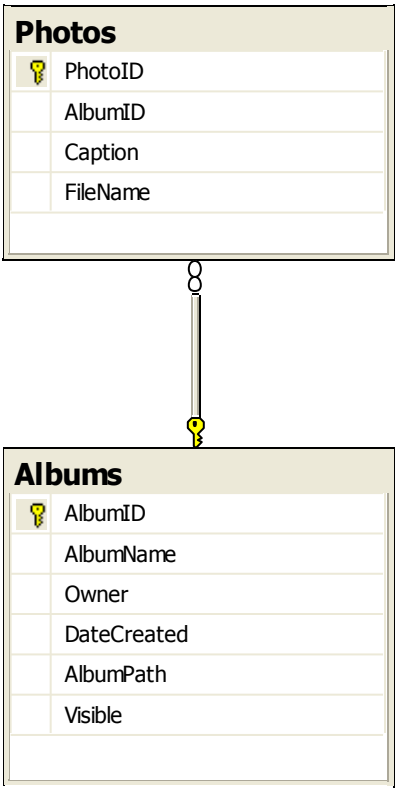
18. *http://samoychiteli.ru/document29256.html* – иллюстрированный самоучитель по языку SQL для начинающих.

19. *http://www.sql.ru/docs/mssql/tsql_ref/* – краткий справочник по языку Transact SQL; сам сайт *http://www.sql.ru* представляет великолепный ресурс по технологиям «клиент-сервер».

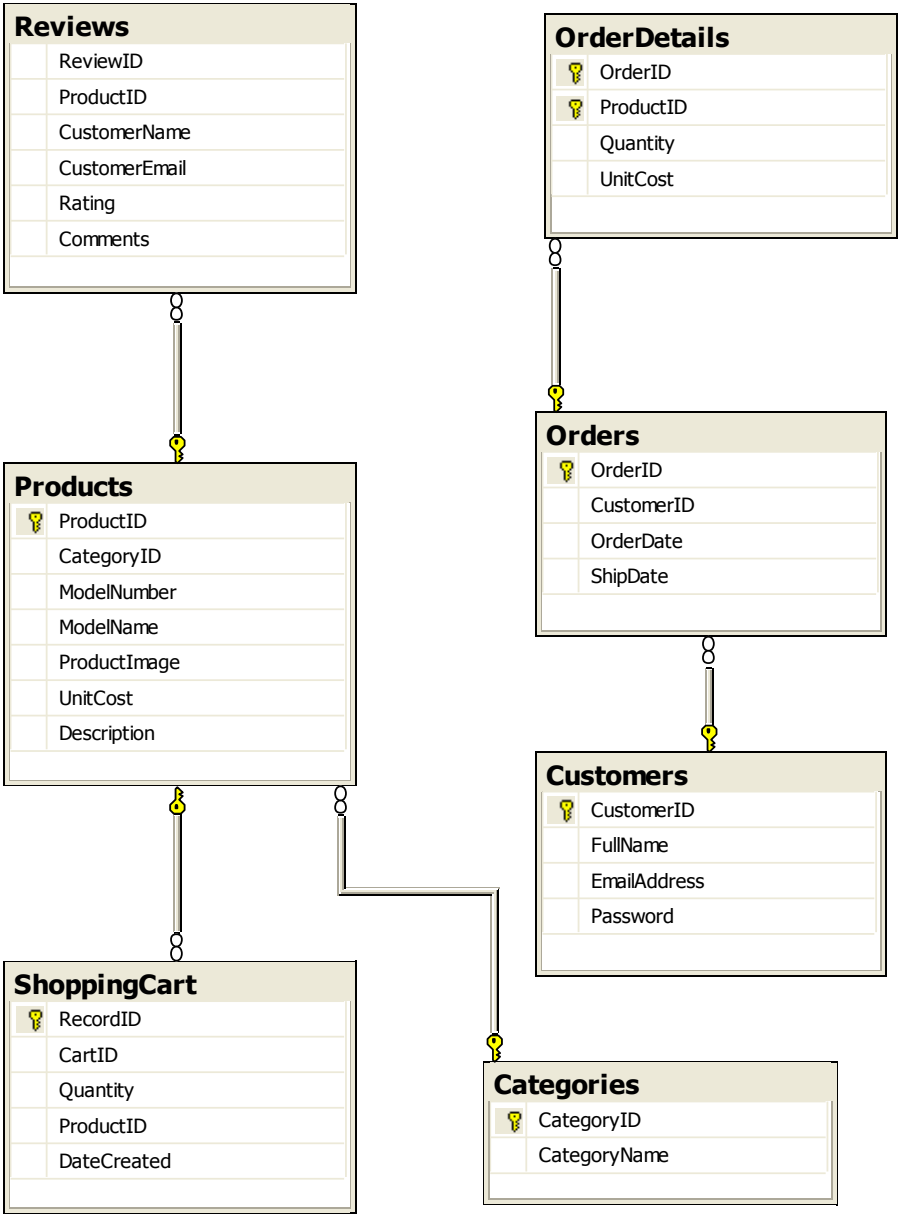
20. *http://www.sql-ex.ru/help/select0.php* – самоучитель по языку SQL; есть довольно много примеров типа «Как это сделать на SQL?».

ПРИЛОЖЕНИЕ
Диаграммы (схемы) баз данных, использующихся в курсе

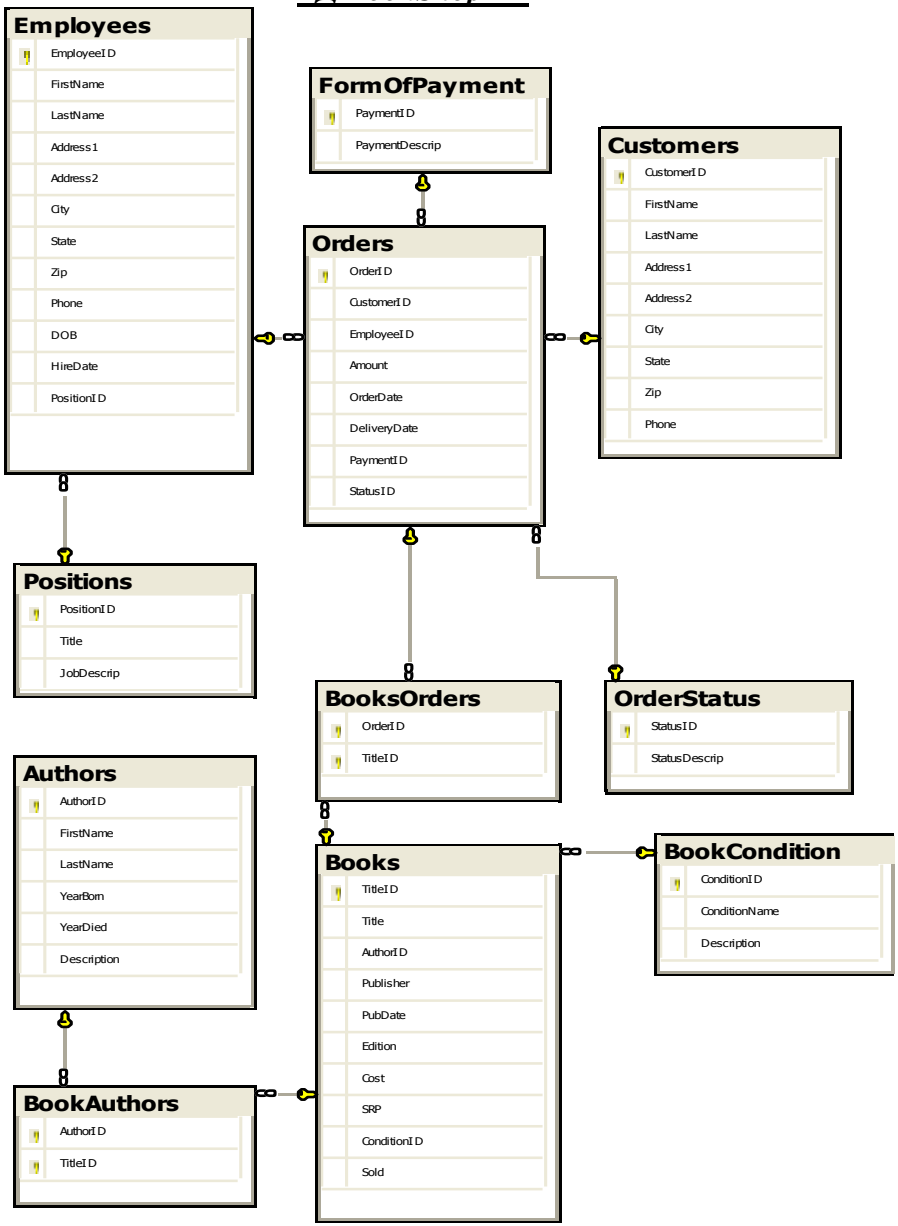
БД Photos



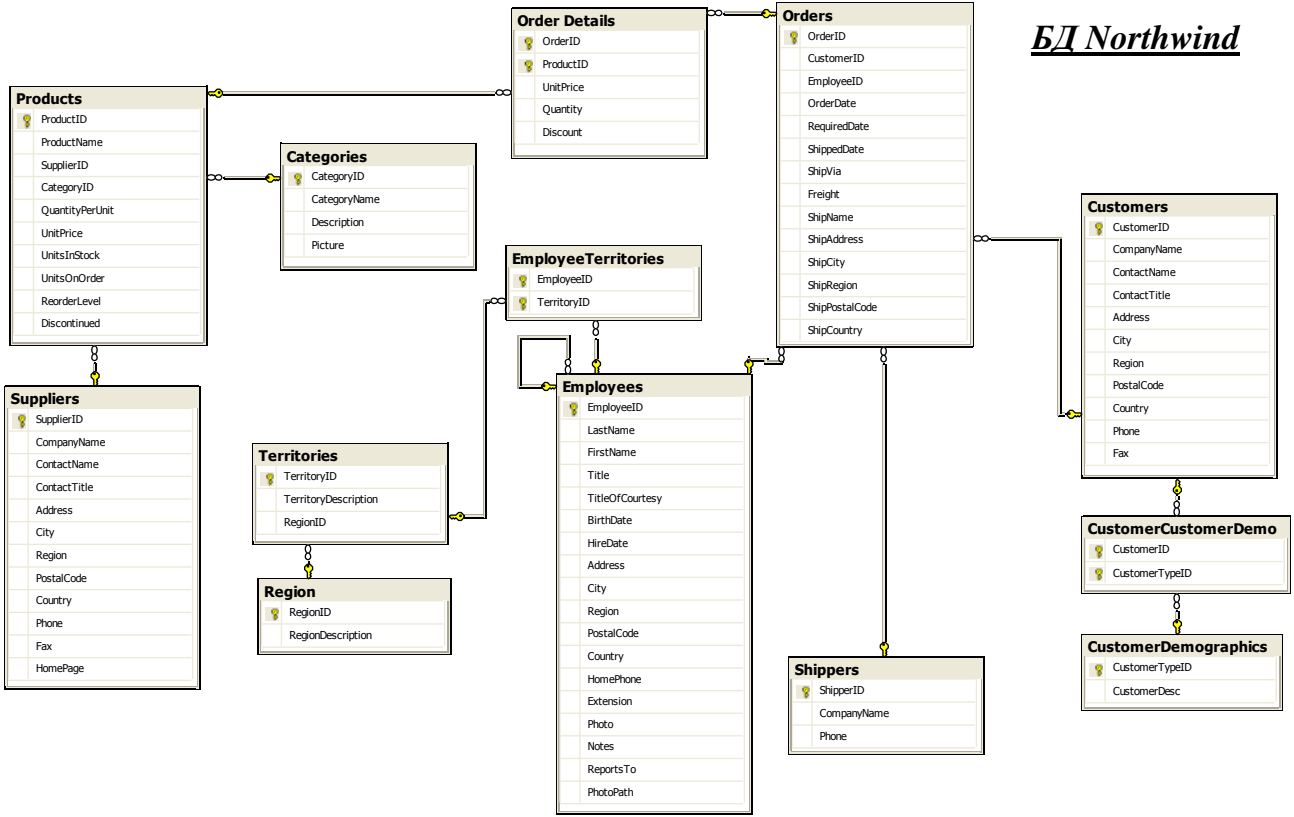
БД Store


Reviews
ReviewID
ProductID
CustomerName
CustomerEmail
Rating
Comments
OrderDetails
OrderID
ProductID
Quantity
UnitCost
Orders
OrderID
CustomerID
OrderDate
ShipDate
Products
ProductID
CategoryID
ModelNumber
ModelName
ProductImage
UnitCost
Description
Customers
CustomerID
FullName
EmailAddress
Password
ShoppingCart
RecordID
CartID
Quantity
ProductID
DateCreated
Categories
CategoryID
CategoryName

БД BookShopDB



БД Northwind



БД GrocerToGo

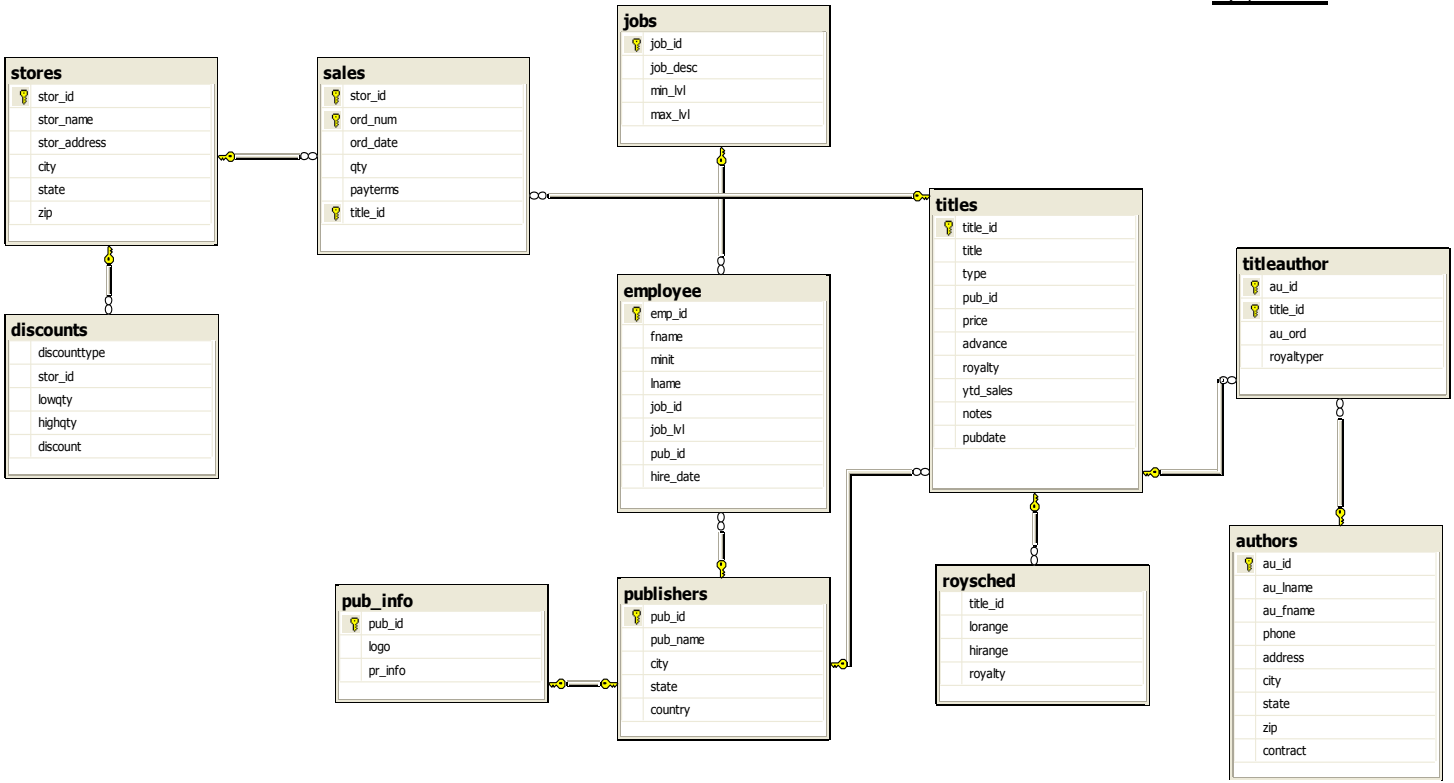
Products
ProductID
CategoryID
ProductName
ProductDescription
UnitPrice
ImagePath
ServingSize
Servings
Quantity
MinOnHand
MaxOnHand
Manufacturer

Customers
CustomerID
FirstName
LastName
Address
City
State
Zip
HomePhone
WorkPhone
Email
Reference
FamilyCount
PetCount
CardNumber
CardType
Expiration
CardName

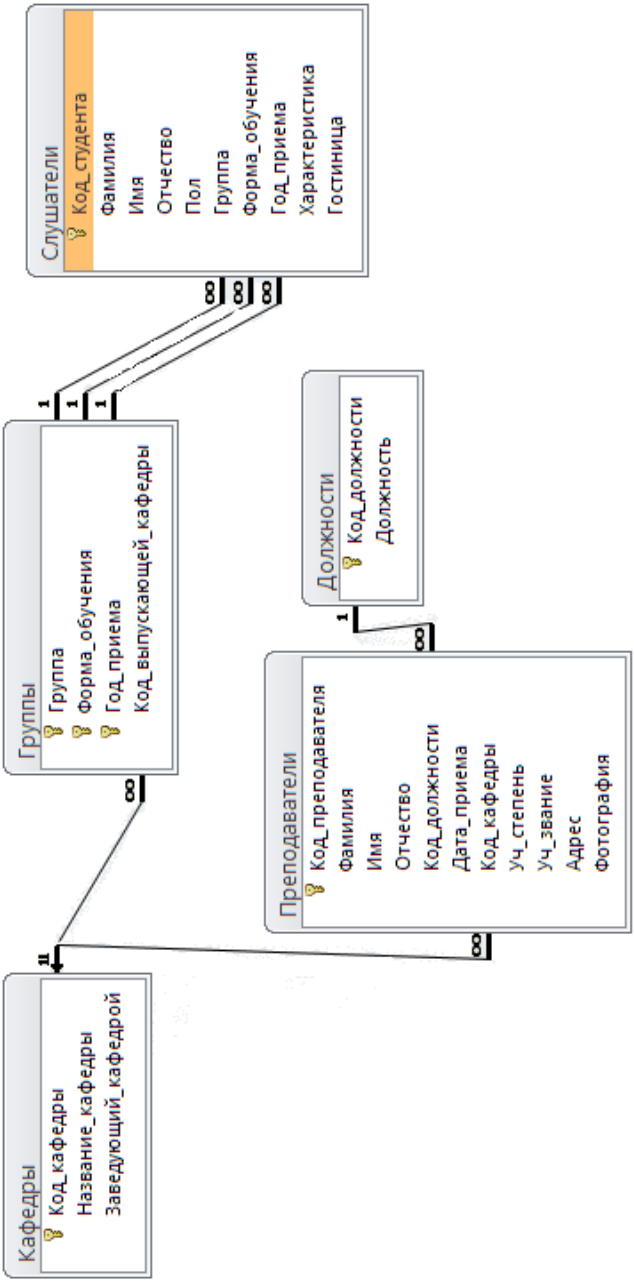
ProductDetails
ProductID
Name
Grams
[Percent]

Categories
CategoryID
CategoryName

БД Pubs


stores
stor_id
stor_name
stor_address
city
state
zip
sales
stor_id
ord_num
ord_date
qty
payterms
title_id
jobs
job_id
job_desc
min_lvl
max_lvl
titles
title_id
title
type
pub_id
price
advance
royalty
ytd_sales
notes
pubdate
titleauthor
au_id
title_id
au_ord
royaltyper
discounts
discounttype
stor_id
lowqty
highqty
discount
employee
emp_id
fname
minit
lname
job_id
job_lvl
pub_id
hire_date
pub_info
pub_id
logo
pr_info
publishers
pub_id
pub_name
city
state
country
roysched
title_id
lorange
hirange
royalty
authors
au_id
au_lname
au_fname
phone
address
city
state
zip
contract

БД study5_08


Слушатели
Код_студента
Фамилия
Имя
Отчество
Пол
Группа
Форма_обучения
Год_приема
Характеристика
Гостиница
Группы
Группа
Форма_обучения
Год_приема
Код_выпускающей_кафедры
Должности
Код_должности
Должность
Преподаватели
Код_преподавателя
Фамилия
Имя
Отчество
Код_должности
Дата_приема
Код_кафедры
Уч_степень
Уч_звание
Адрес
Фотография
Кафедры
Код_кафедры
Название_кафедры
Заведующий_кафедрой

Учебное издание

Александр Иванович Митин

Работа с базами данных Microsoft SQL Server

Сценарии практических занятий

Издательство «Директ-Медиа»
117342, Москва, ул. Обручева, 34/63, стр. 1
Тел/факс + 7 (495) 334-72-11
E-mail: manager@directmedia.ru
www.biblioclub.ru

Отпечатано в ООО «ПАК ХАУС»
142172, г. Москва, г. Щербинка,
ул. Космонавтов, д.16