В.В. Братищенко

Реляционные и документационные базы данных

Учебное пособие

Министерство науки и высшего образования Российской Федерации
Байкальский государственный университет

В.В. Братищенко

Реляционные и документационные базы данных

Учебное пособие

Иркутск
Научное издательство БГУ
2020

УДК 681.3
ББК 32.973
Б87

Печатается по решению редакционно-издательского совета
Байкальского государственного университета

Рецензенты канд. физ.-мат. наук, доц. В.В. Ступин
канд. техн. наук, доц. А.В. Родионов

Братищенко В.В.
Б87 Реляционные и документационные базы данных : учеб. пособие / В.В. Братищенко. – Иркутск : Науч. изд-во БГУ, 2020. – 130 с. – URL: http://lib-catalog.bgu.ru.

Содержит основные сведения о базах данных и системах управления базами данных. Изложены общетеоретические вопросы моделирования данных с применением модели «сущность-связь», теоретические основы реляционной модели, технология проектирования с применением теории нормальных форм отношений. Рассмотрены наиболее распространенные реляционные базы данных и практическое использование MS SQL сервера для создания, использования и администрирования реляционной базы данных. Описан язык SQL: команды корректировки и выбора данных, построение и использование программных объектов. Приведены сведения о технологии многомерного анализа данных (OLAP – On-Line Analytical Processing): структуры хранилищ данных, основные операции многомерного анализа, язык MDX-запросов к многомерным данным. Рассмотрена практическая работа по созданию и использованию OLAP-технологии на основе Microsoft Analysis Services. В качестве документационной базы приведены сведения о работе с системой MongoDB: структуры данных, команды создания и изменения структур, извлечения и обработки данных.

Для студентов, обучающихся по направлениям бакалавриата 09.03.03 Прикладная информатика и 38.03.05 Бизнес-информатика.

УДК 681.3
ББК 32.973

Оглавление

Введение

Базы данных являются основным средством хранения структурированных данных большого объема. Широкое применение вычислительной техники для решения задач управления привело к необходимости использования единого хранилища данных, в которое загружались бы данные из разных источников и которое бы обеспечивало информационные потребности большого количества пользователей. Единое хранилище обеспечивает очень важный принцип «однократного ввода». Принцип важен не только для экономии усилий по вводу данных, но и для исключения нескольких версий, которые неизбежно (человеческий фактор!) приводят к рассогласованию данных.

Другая важная особенность технологии баз данных – это коллективный доступ к данным. Коллективный доступ подразумевает решение вопроса о разделении полномочий по вводу, корректировке и обработке данных. В хорошо работающей информационной технологии практически не возникает коллизий – каждый пользователь имеет строго определенные полномочия: разные операционисты выполняют регистрацию строго определенного, разного контента, все данные в совокупности описывают протекание большого количества бизнес-процессов, лица, принимающие решения, получают сводную информацию, необходимую и удобную для обеспечения принятия верных решений. Каждый вводит и получает данные в соответствии со служебными обязанностями и потребностями.

В технологическом плане каждая база данных имеет базовые структуры, обеспечивающие хранение и обработку данных. Для реляционных баз это таблицы специальной структуры, для документационных – коллекции документов, состоящие из компонентов, графовые базы хранят данные в виде графов и т.д. Какая бы информация ни хранилась в базе, она будет представлена в виде базовых структур. Чем «жестче» структура, тем выше эффективность обработки и больше возможностей управления доступом. С другой стороны, «жесткие» структуры существенно повышают трудоемкость адаптации и развития.

Базовые структуры определяют язык обработки: команды используют базовые структуры для получения результата, а результат снова представляется в виде базовой структуры. Для реляционных баз данных такой язык предоставляет операции по объединению данных из разных таблиц и преобразованию результатов обработки в табличную форму.

Реляционные базы являются, по-видимому, наиболее «жесткими». В них любое изменение структуры таблиц влияет на логику хранения и обработки данных. Например, товар описывается набором атрибутов – одна строка таблицы описывает один товар. Если нужно добавить новый атрибут некоторого товара, то появляется новый столбец и, соответственно, новый атрибут у всех товаров.

Документационные базы избавлены от такого недостатка: любой документ может иметь компонент, которого нет в других документах. Замечательно! Но представим вычисление предоплаты по коллекции документов, в которых для этого используются компоненты, названные по-разному или вовсе от-

сутствующие. Или представьте правило, определяющее права доступа к таким компонентам. Чем больше гибкости в структурах данных, тем сложнее управление и обработка данных. Независимо от степени гибкости структуры любой базы нуждаются в тщательном проектировании с учетом особенностей ввода и использования данных.

Кроме собственно структур с данными, любая база данных содержит описание используемых структур – так называемы метаданные. Например, в реляционных базах метаданные включают списки таблиц и колонок таблиц с указанием имен, типов и свойств. Метаданные позволяют указывать в командах имена структур и компонентов данных. Это радикально меняет технологию работы с данными: непосредственный доступ к данным в файлах заменяется на передачу команд специальной программе – системе управления базами данных (СУБД), которая выполняет команду и возвращает результат. В целом такая технология и позволяет добиться нужного результата – совместного эффективного накопления и использования данных в строгом соответствии с полномочиями.

Использование серверов баз данных позволяет распределить работу между различными вычислительными процессами, которые могут выполняться одновременно: сервер баз данных (СУБД) выполняет запросы на извлечение данных, сервер приложений выполняет обработку, интернет-сервер передает данные по сети, клиентский процесс обеспечивает интерфейс с пользователем.

В современных системах данных может быть так много, что они физически не могут быть размещены на одном сервере. С другой стороны, применение для работы с данными нескольких серверов позволяет выполнять большую часть обработки данных более эффективно. Распределенное хранение данных предполагает возможность распределенной обработки данных. Каждый сервер обрабатывает свои данные и некоторый узел объединяет результаты их работы. Такие процедуры становятся составной частью технологии «больших данных».

1. Общие сведения

1.1. Причины и предпосылки появления баз данных

Базы данных на сегодняшний день являются наиболее распространенным способом хранения информации. Как форма хранения они обеспечивают быстрое проектирование информационной системы, достаточный уровень независимости программ и данных, надежность хранения, безопасность и защиту от несанкционированного доступа, разделение прав доступа для пользователей базы данных, эффективное выполнение запросов.

Уже первые применения вычислительной техники для решения задач управления привели к необходимости хранения на машинных носителях (в электронной форме) больших объемов информации. Жесткая привязка программ и структур данных приводила к необходимости переписывать программы при любых изменениях в форматах данных. Поэтому следующее поколение систем разрабатывалось по принципу «независимости программ от данных». Для этого необходимо хранить кроме данных еще и описание структур хранения – так называемые метаданные (данные о данных). Специальный компонент программного обеспечения использует метаданные для записи и извлечения данных из хранилища.

Распространение автоматизированных технологий управления привело к использованию разными подразделениями одной и той же информации. Хранение нескольких электронных версий данных вызвало помимо дублирования еще и появление и накопление противоречий в разных версиях. Решением этой проблемы стало совместное использование разными пользователями одной и той же коллекции данных – базы данных.

Таким образом, обеспечение независимости программ и данных, с одной стороны, и совместного использования данных – с другой и привели к возникновению понятия ***базы данных (БД)*** – специальным образом организованной системы хранения данных и к созданию ***системы управления базами данных (СУБД)*** – программы, обеспечивающей доступ к базе для выполнения запросов на корректировку и извлечение данных. Широкое применение баз данных привело к появлению технологии Information engineering [8] построения информационных систем, в которой центральным ядром системы становятся данные (базы данных). В таких системах пользователи получают доступ к базе данных в соответствие со служебными полномочиями при помощи приложений и посредничестве СУБД.

Известно несколько определений базы данных.

База данных – поименованная, целостная, единая система данных, организованная по определенным правилам, которые предусматривают общие принципы описания, хранения и обработки данных [5].

База данных есть совокупность данных, организованных в соответствии с некоторой концептуальной моделью данных, которая описывает характеристики этих данных и взаимоотношения между соответствующими им реалиями, и которая предназначена для информационного обеспечения одного или нескольких приложений [4].

При всей разнице общим в этих определениях является концептуальное единство. Это реализовано в виде использования однотипных структур хранения (например, все данные хранятся в таблицах) и применения соответствующих команд изменения и извлечения данных. Причем извлекаются данные в виде все тех же базовых структур хранения данных.

Система управления базами данных (СУБД) – это программа (комплекс программ), которая обеспечивает хранение данных и доступ к базам данных. СУБД выполняет следующие функции:

– создание информационных структур, описание и поддержание целостности;

– выполнение запросов;

– поддержка многопользовательского режима;

– обеспечение надежности и безопасности хранения;

– управление полномочиями пользователей;

– выполнение сервисных функций (экспорт-импорт, резервное копирование и восстановление).

СУБД реализует общие принципы описания данных в виде ЯОД – языка описания данных (DDL – Data Definition Language) и принципы обработки данных в виде ЯМД – языка манипулирования данными (DML – Data Manipulation Language).

В соответствии с концепцией Information Engineering именно данные, а не функции являются наиболее стабильной и системообразующей составляющей информационной системы. Следовательно, наиболее значимым становится проектирование системы хранения данных. В проектировании информационного обеспечения традиционно выделяют два этапа проектирования:

1. Инфологическое проектирование – решение вопросов, связанных со смысловым содержанием данных:

– о каких объектах и явлениях следует накапливать информацию;

– какие их характеристики и взаимосвязи будут учитываться.

2. Даталогическое проектирование – решение вопросов, связанных с представлением данных:

– типы и форматы данных;

– методы преобразования и смысловой интерпретации данных.

1.2. Модель «сущность-связь»

Наиболее распространенной основой инфологического проектирования является модель «сущность-связь» [2; 3]. В отличие от структурной информационной модели, основанной на разбиении всех данных на составляющие вида структура (набор данных разных типов) или массив (набор данных одного типа), модель «сущность-связь» ориентирует проектировщика на выявление сущностей (наборов однотипных объектов или явлений, о которых необходимо сохранять информацию) и устойчивых связей между ними, необходимых для обеспечения целостности. Например, выделение связи договора с поставщиком

означает, что в договоре указан поставщик и сведения о нем должны быть зарегистрированы в БД.

Инфологическое проектирование начинается с определения ***предметной области*** – части реально существующего мира, состояние которой БД должна адекватно описывать. Как уже упоминалось, ***модель «сущность-связь»*** описывает предметную область в виде множества сущностей и связей между ними. Это описание отображается в виде **диаграмм «сущность-связь»** (Entity-Relationship Diagram – ERD). Для диаграмм «сущность-связь» применяется несколько систем обозначений. Мы будем использовать нотацию Мартина.

Сущность – некоторая часть реального мира, которую можно выделить как самостоятельное целое, взаимодействующее с другими частями. Сущность в модели является обобщенным представителем множества объектов или явлений – экземпляров сущности. Сущности обычно именуются существительными. Например, для регистрации торговых операций можно выделить следующие сущности: продавцы, покупатели, товары. При этом сущность «продавец» представляет в модели общие свойства всех продавцов. Деление на сущности достаточно произвольно. Например, можно объединить продавцов и покупателей в одну сущность – организации. Все объекты реального мира, соответствующие некоторой сущности, в информационной системе будут иметь одинаковый набор описателей – ***атрибутов (полей, реквизитов)***. Например, каждый продавец описывается наименованием, адресом, счетом в банке и т.д. На диаграмме сущность изображается прямоугольником (рис. 1). Таким образом, сущность – это своего рода шаблон для описания определенного объектов одного вида.

Сущности можно классифицировать следующим образом:

- роли (люди, организации);
- реальные предметы;
- события (договора, поставки, оплаты);
- расположения.

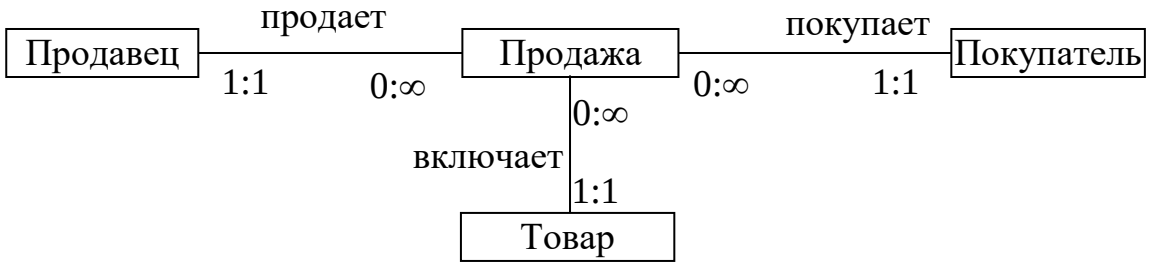


Рис. 1. Модель «сущность-связь» для торговых операций

Связь – это некоторая характеристика взаимодействия или ассоциации сущностей. Связь обычно именуется глаголом. Так же как сущность, связь является обобщенным представителем экземпляров связей. На диаграмме связь изображается линиями, который соединяют связываемые сущности (см. рис. 1).

Наиболее распространенным видом связи является бинарный – когда сущности связаны попарно. Например, «банк – клиент». Если выделена связь

нескольких сущностей, то, скорее всего, в виде связи выделено некоторое явление. Например, «продажу» можно рассматривать как связь «продавца», «товара» и «покупателя». Такие связи имеют набор атрибутов (цена, количество) и должны быть преобразованы в сущности (как это сделано на рис. 1) с бинарными связями с другими сущностями.

Связь обычно характеризует ссылочную целостность и не имеет атрибутов. Вместо атрибутов связь характеризуют количеством экземпляров связываемых сущностей. Например, у продажи один и только один продавец, а у продавца может быть несколько продаж (ноль или много). Различают связи «один к одному», «один ко многим» и «многие ко многим». Точнее, множество экземпляров каждой из связываемых сущностей описывают двумя числами: ординальное число указывает минимальное количество экземпляров сущности в связи, кардинальное – максимальное. Для связи «продажа включает товар» на рис. 1 установлено, что продажа включает один и только один товар, а товар включается в ноль или несколько продаж.

Связи «многие ко многим», так же как связи более двух сущностей, требуют описания набором атрибутов, поэтому такие связи также преобразуют в некоторую сущность. Связь преподавателей с учебными группами можно описать в виде сущности «занятие», которое имеет две связи: с преподавателем и с группой.

Для сущности можно установить ***ключ*** – множество атрибутов, которое позволяет однозначно идентифицировать один экземпляр сущности. Ключей может быть несколько. Например, работника можно идентифицировать по серии и номеру паспорта, или по табельному номеру, или по индивидуальному номеру налогоплательщика. Один ключ из возможных выбирают для использования в качестве ссылки и называют ***первичным ключом.*** Когда надо сослаться на экземпляр сущности, используют соответствующее значение первичного ключа.

Рассматривая связь, можно выделить независимую (родительскую) сущность – существующую независимо от другой и зависимую – дочернюю, не существующую без родительской сущности. На рис. 1 независимыми сущностями являются покупатели, продавцы и товары, а зависимой – продажа. Дочерняя сущность содержит ссылку на соответствующую родительскую сущность. Эта ссылка представляет собой набор атрибутов первичного ключа родительской сущности. Ссылочная целостность заключается в правильности значений всех ссылок – не допускаются ссылки на несуществующие экземпляры сущности. Например, код продавца в договоре является ссылкой на описание продавца. Ссылочная целостность заключается в обязательном существовании в БД описания продавца с указанным кодом. Не допускается также существование двух описаний продавцов с одинаковым кодом.

Сущности могут быть связаны попарно несколькими связями. Например, сущность «Валюта» может быть связана с сущностью «Обмен» двумя связями: «продается» и «покупается». Это означает неоднократное употребление ссылок с разными смыслами. «Обмен» должен ссылаться и на продаваемую валюту, и на покупаемую валюту.

Возможен случай рекурсивной связи. Например, сущность «сотрудник» может быть связана сама с собой связью «подчиняется» (описание каждого со-

трудника содержит ссылку на другого сотрудника – непосредственного начальника). Рекурсия может быть непосредственной – сущность связывается сама с собой, как в рассмотренном примере, или опосредованной, если рекурсивная цепочка связей замыкается через другие сущности. Например, сущность «сотрудник» связана с сущностью «подразделение» связями «является руководителем» и «работает в», которые образуют рекурсивную связь (описание сотрудника содержит ссылку на подразделение, описание подразделения ссылку на сотрудника – начальника подразделения).

Связи могут отражать классификацию сущностей. Для этого вводят сущность – ***супертип***, которая объединяет характерные черты нескольких сущностей – ***подтипов*** (рис. 2). Например, роль супертипа может играть сущность «организация» по отношению к подтипам «продавец» и «покупатель». Соединение супертипа с подтипами на диаграмме образует категориальную связь, которая отображается на диаграмме линиями, при этом для выделения супертипа используется черточка.

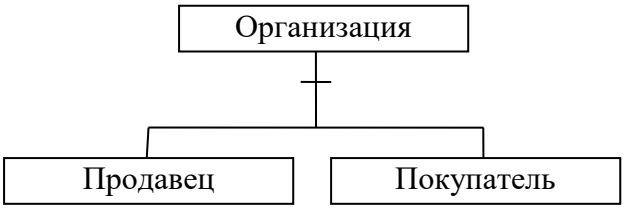


Рис. 2. Пример супертипа и подтипов

Рекомендуется следующая последовательность построения модели «сущность-связь»:

1. Идентификация сущностей. Для идентификации полезными оказываются следующие приемы:

– при интервьюировании пользователей выделяйте ключевые слова;

– просите пользователей определить объекты и явления, о которых необходимо собирать, хранить и извлекать информацию;

– изучайте формы документов;

– исследуйте уже созданные и применяемые ИС.

Выясните количество экземпляров сущностей (десятки, сотни...). Придумайте простые звучные содержательные имена, имеющие точный смысл. Старайтесь избегать аббревиатур. Имена должны быть уникальны.

2. Идентификация атрибутов. Для этого выделяют свойства, характеристики и другие описатели сущностей. При этом важно, чтобы названия максимально точно отражали понятия (например, не просто «Дата», а «Дата поставки»).

3. Описание каждой сущности набором атрибутов. При этом следует избегать включения в описание одной сущности атрибутов другой сущности, например, в описание поставщика не следует включать номер договора. Также не рекомендуется использовать атрибуты, являющиеся списками, например атрибут «иностранные языки, которыми владеет работник».

4. Выбор первичных ключей. Значения первичного ключа используют для ссылки. Поэтому стараются выбирать первичные ключи минимальной длины для экономии памяти и увеличения скорости поиска по ключу. Не рекомендуется выбирать в качестве первичных ключей описатели, которые могут изменяться. Например, если выбрать серию и номер паспорта в качестве первичного ключа работника, то при замене паспорта придется менять в базе все ссылки на данного работника. Обычно в качестве первичного ключа выбирают внутренний системный номер экземпляра сущности – код, который формируется автоматически при добавлении данных.

5. Построение диаграммы. В сложных случаях формируют несколько диаграмм, каждая из которых описывает определенную область учета. Например, в БД «Контингент студентов» одна диаграмма описывает деление контингента на факультеты, специальности, группы, другая представляет структуры данных для учета успеваемости и т.д.

Важной частью работы по формированию модели данных всей информационной системы, объединяющей несколько подсистем, является согласование подмоделей данных (подсхем) с моделью данных организации в целом (рис. 3). Для этого используются следующие операции:

1. Установление идентичности. Два элемента идентичны, если они имеют одинаковое смысловое значение.

2. Агрегация – объединение связанных элементов модели в один сложный (адрес, банковские реквизиты).

3. Обобщение – разбиение множества сущностей на классы эквивалентности и введение супертипов – сущностей, соответствующих понятию класса. Таким образом, создается иерархия сущностей.

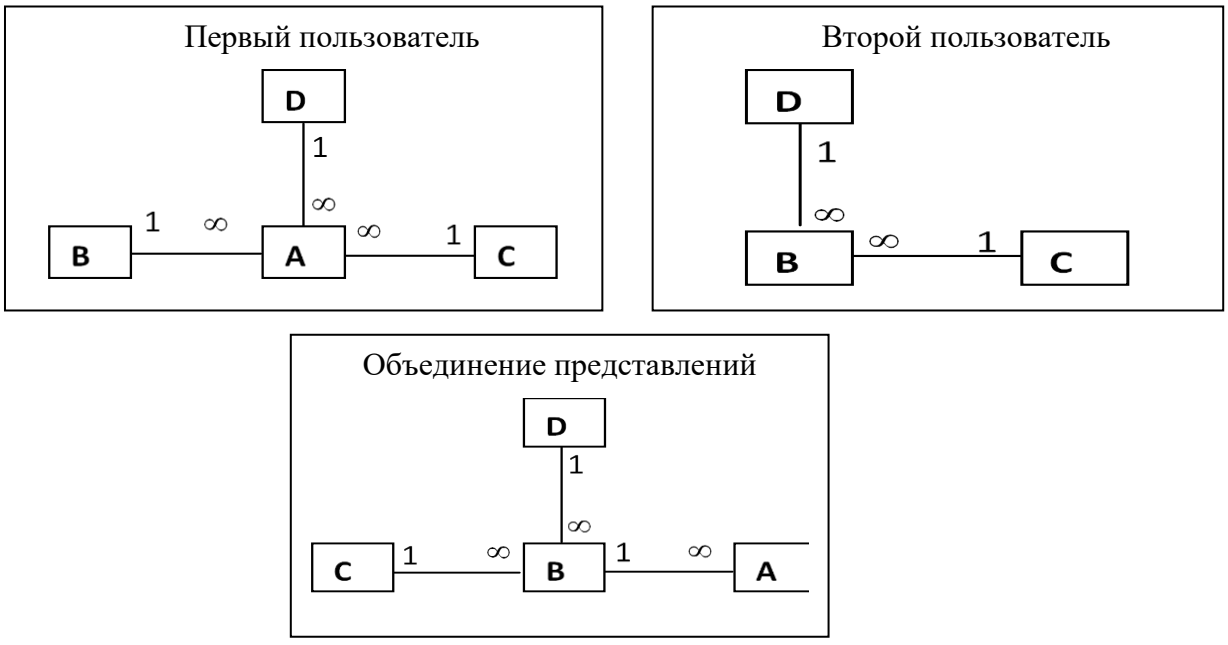


Рис. 3. Объединение представлений пользователей

Проблема заключается в использовании разными пользователями разных терминов и описаний для одних и тех же понятий.

1.3. Бизнес-правила

Технология использования БД включает процедуры изменения и использования данных при выполнении бизнес-функций. Для каждой функции указывается должностное лицо, ответственное за выполнение функции, основания ее выполнения (документы, распоряжения, наблюдения, события), регламент выполнения функции, условия выполнения функции, особенности выполнения функции, правила регистрации данных, вытекающие из правил соответствующего учета. Все это объединяют в понятие «бизнес-правила». Например, в бухгалтерском учете для выполнения регистрации хозяйственной операции необходим соответствующим образом оформленный первичный документ, для которого указываются правила проводки и должностные лица, имеющие полномочия выполнять регистрацию.

Технологию работы с БД можно описывать в виде потока событий, приводящих к изменениям в БД и выборкам из БД. Анализ событий заключается в выделении реальных экономических событий, которые могут происходить в предметной области, операций с данными, которые вызывает каждое событие, и обстоятельств, влияющих на выполнение операций. На товарно-сырьевой бирже такими событиями могут быть регистрация нового участника торгов, заявки на продажу или покупку, заключение сделки, оплата товара. Для каждого события перечисляются сущности и операции (выборка, добавление, изменение, удаление данных), которые события вызывают. В обстоятельства выполнения операций включают должностное лицо, ответственное за внесение данных и правила выполнения операций.

Рекомендуется следующая последовательность для проведения анализа событий:

1-й шаг. Идентификация событий для независимых сущностей. Результатом идентификации является таблица, состоящая из следующих колонок:

- имя события;
- краткое описание события;
- сущность, затрагиваемая событием;
- тип действия: добавление, изменение, удаление или выборка данных;
- условия выполнения действия.

Одно событие может влиять на много сущностей. Некоторые события могут обуславливаться более чем одни условием. Идентификация событий требует привлечения пользователей.

Условия можно разбить на два класса:

- вытекающие из экономической практики (известны пользователям);
- обусловленные связями структур данных (определяются системными аналитиками): оцениваются связи двух сущностей – выделяется независимая, родительская сущность и зависимая, дочерняя сущность. Для корректного добавления экземпляра дочерней сущности должен существовать экземпляр родительской. Удаление экземпляра родительской сущности возможно, если нет связанных с ним экземпляров дочерней.

Для событий, вызывающих изменения, должны указываться атрибуты, которые могут изменяться. Затем выполняется идентификация событий для зависимых сущностей.

2-й шаг. Консолидация общих событий. Проверяется таблица – выделяются одинаково названные разные сущности (события) и по-разному названные одинаковые. После этого выполняется реорганизация таблицы – для каждого события указываются сущности и действия.

По результатам построения модели данных и анализа событий пересматривается описание потоков данных. При этом возможно:

– добавление входных данных для проверки условий (например, на существование родителя);

– добавление атрибутов;

– добавление выходных данных;

– добавление новых операций (например, проверок).

Описание событий (технологии работы с БД) служит нескольким целям. Во-первых, можно проверить жизненный цикл каждой сущности: данные как минимум должны добавляться и использоваться, возможны также операции корректировки и удаления. Во-вторых, привязка процедур по рабочим местам позволяет определить полномочия на уровне базы данных каждого должностного лица, а также список приложений и функциональные возможности каждого приложений. В-третьих, создается описание бизнес-правил, которые становятся частью базы данных.

Вопросы

1. Приведите причины возникновения баз данных.
2. Что такое метаданные?
3. Дайте определение базы данных и системы управления базами данных.
4. Перечислите основные функции СУБД.
5. Перечислите и укажите назначение представлений данных в БД.
6. В чем заключается концепция Information Engineering?
7. Определите содержание инфологического и даталогического проектирования.
8. Дайте определение сущности и связи.
9. Укажите, чем характеризуются сущности и связи.
10. Перечислите виды связей. Какие связи обычно отображают модели «сущность-связь» и почему?
11. Определите понятия «ключ» и «первичный ключ».
12. Как выделяются «родительские» и «дочерние» сущности?
13. В чем заключается ссылочная целостность данных?
14. Опишите последовательность построения модели «сущность-связь».
15. Что входит в технологию использования БД?
16. Приведите два-три примера бизнес-правил, которые необходимо выполнять для поддержания целостности БД.
17. Опишите технологию анализа событий.

2. Реляционная модель данных

2.1. Описание реляционной модели данных

Основы реляционной модели данных [2] были сформулированы Э.Ф. Коддом в 1970 г. Структурой хранения данных в этой модели является отношение – таблица со следующими свойствами:

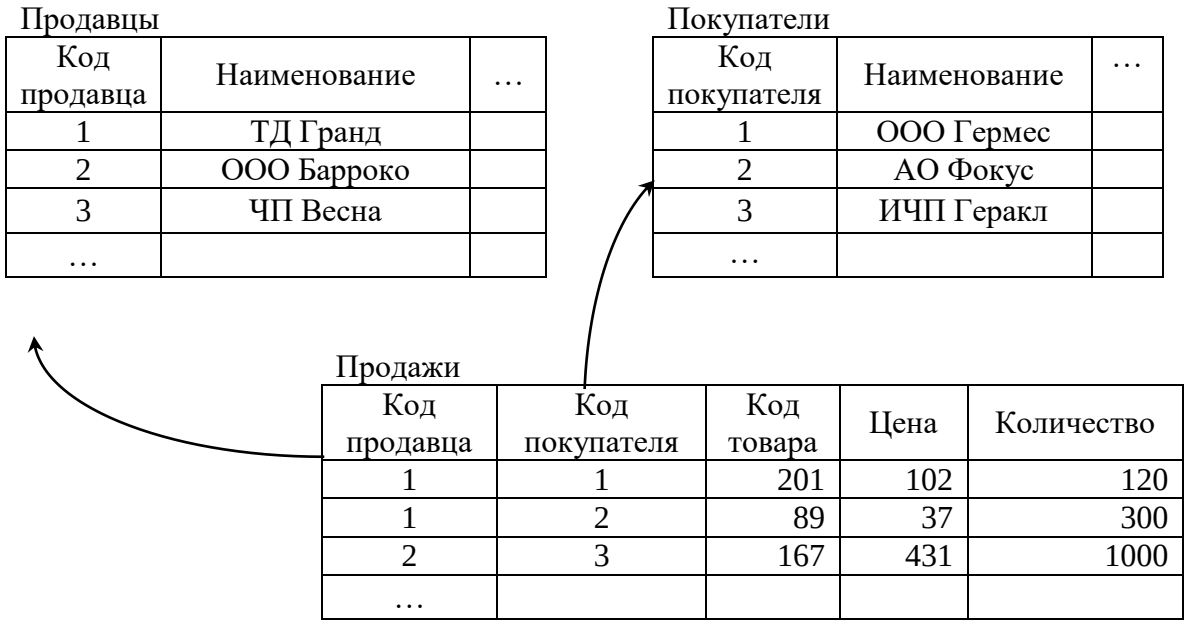
1. Каждый столбец содержит информацию одного типа.

2. Ячейки – поля – таблицы не содержат агрегатов (структур или массивов) данных.

3. Таблицы не содержат одинаковых строк.

4. Порядок строк и столбцов не имеет значения. Все операции используют содержательную сторону данных, а не их расположение внутри таблицы.

Для описания связей вводятся первичные ключи – минимальные наборы полей, позволяющие указывать ровно одну строку (кортеж) таблицы. Значение ключа используется для ссылки в других таблицах, что и является описанием связей данных. Поскольку первичный ключ играет ведущую роль в описании связей и поиске данных, размер ключа стараются сделать минимальным для оптимизации поиска. Это приводит к использованию номеров или кодов в качестве первичных ключей. На рис. 4 приведено представление данных о продажах в реляционной модели. Стрелки на рисунке отражают ссылки данных одной таблицы на данные других таблиц. В следующем параграфе приведено формальное описание реляционной модели данных.



Стрелками показаны связи полей таблиц

Рис. 4. Представление данных о продажах в реляционной модели данных

Для запросов к реляционным данным придумано несколько языков. Все они используют похожие операции в разных нотациях. Результатом каждой

операции является отношение. Кроме операций, над таблицами как над множествами записей вводится операция соединения, позволяющая из нескольких таблиц получить таблицу-результат, операция проекции для выделения таблицы с нужным набором полей (колонок), операция фильтрации для выбора записей (строк), удовлетворяющих некоторому условию и ряд других. Язык реляционной алгебры и исчисление отношений описаны в 2.2 и 2.3, а язык SQL – в главе 4.

В реляционной модели данные представлены в виде набора отношений (relation). ***Отношение*** $r = \{(d_1, ..., d_n), d_1 \in D_1, ..., d_n \in D_n\}$ есть множество ***кортежей*** $(d_1, ..., d_n)$, значения которых принадлежат ***доменам*** $D_1, ..., D_n$. Таким образом, отношение есть подмножество декартова произведения доменов $r \subset D_1 \times ... \times D_n$.

Число n называют ***степенью отношения***. Количество кортежей в отношении r называют ***мощностью отношения*** r.

Отношение является аналогом таблицы с приведенными в 3.3 свойствами. Колонки таблицы делят каждый кортеж (запись) на ***атрибуты*** – компоненты кортежа. Для указания атрибута используют уникальные в рамках одной таблицы имена. Термин «атрибут» используется в разных смыслах. Во-первых, атрибут определяет множество возможных значений соответствующей колонки таблицы, так как областью значений атрибута является соответствующий домен. Во-вторых, атрибут используется в программах для описания доступа к соответствующей компоненте кортежа – аналогичным понятием является ***поле*** записи. Атрибут A_i – это функция, вычисляющая значение i-й компоненты кортежа отношения r. Множеством значений атрибута A_i является домен D_i. В алгоритмических языках атрибут обозначается чаще именем и реже номером атрибута в кортеже.

Схемой $r(R)$ отношения r называют множество атрибутов $R = \{A_1, ..., A_n\}$. Реляционная модель данных или схема БД – это множество схем отношений, дополненное различными ограничениями и правилами (бизнес-правилами).

Далее по тексту используются следующие обозначения:

– $t[A_i]$ – значение атрибута A_i в кортеже $t \in r$ отношения r;

– $t[X] = (t[A], t[B], ...)$ – X-значение кортежа t на множестве атрибутов $X = \{A, B, ...\}$.

Важное значение в теории реляционных БД имеет ключ, позволяющий выделять ровно один кортеж из отношения. Ключом K отношения $r(R)$ называют множество атрибутов $K = \{B_1, ..., B_m\} \subset R$ такое, что в отношении r не существует двух кортежей с одинаковым K-значением ключа и ни одно собственное подмножество ключа K не обладает этим свойством.

2.2. Операции над отношениями. Реляционная алгебра

Над отношениями r и s с одинаковыми схемами можно выполнять теоретико-множественные операции:

1. ***Объединение отношений*** – множество кортежей, принадлежащих отношению r или отношению s:

r **UNION** $s = \{ t : (t \in r) \cup (t \in s) \}$.

2. ***Пересечение отношений*** – множество кортежей, принадлежащих ***r*** и ***s*** одновременно:

***r* INTERSECT *s* = { *t* : (*t* ∈ *r*) & (*t* ∈ *s*) }.**

3. ***Разность отношений*** – множество кортежей, принадлежащих ***r*** и не принадлежащих ***s***:

***r* EXCEPT *s* = { *t* : (*t* ∈ *r*) & (*t* ∉ *s*) }.**

4. ***Дополнение отношения r*** можно определить как разность множества всех возможных записей отношения ***r*** и самого отношения ***r***:

NOT *r* = ($D_1 \times D_2 \times \ldots \times D_n$) EXCEPT *r* = { *t* : (*t* ∈ $D_1 \times D_2 \times \ldots \times D_n$) & (*t* ∉ *r*) }.

За исключением тривиальных случаев мощность дополнения выражается огромным числом, и выполнение этой операции физически невозможно. Для уменьшения количества получаемых кортежей вводят операцию «активное дополнение».

5. ***Активное дополнение***. Вместо декартова произведения доменов в этой операции используют декартово произведение множеств используемых значений $D'_i = \{ t[A_i] : t \in r \}$.

Кроме множественных операций, вводят специфические операции для манипулирования отношениями.

6. ***Выбор*** или ***фильтрация*** – множество кортежей, обращающих логическую формулу $\theta(t)$ в истину:

***r* WHERE $\theta(t)$ = { *t* : $\theta(t)$ }.**

7. ***Проекция*** – из отношения вычеркиваются столбцы, не входящие в множество атрибутов ***X*** и повторяющиеся строки:

***r*[*X*] = { *t*[*X*]: *t* ∈ *r* }.**

8. ***Естественное соединение***

***q* (*R* ∪ *S*) = *r* (*R*) JOIN *s* (*S*)**

= { *t* : (∃ t_r ∈ *r* : (∃ t_s ∈ *s*: (t_r = *t* [*R*]) ∩ (t_s = *t* [*S*])

получается как результат комбинирования каждого кортежа t_r отношения ***r*** с каждым кортежем t_s отношения ***s*** при условии совпадения одноименных атрибутов $t_r[R \cap S] = t_s[R \cap S]$ (Рис. 5).

Для различных прикладных аспектов важно разлагать при помощи проекции отношение на множества отношений, содержащих туже самую информацию. Отношение ***q* (*R* ∪ *S*)** разложимо без потерь на отношения ***r* (*R*) = *q* [*R*]** и ***s* (*S*) = *q* [*S*]** если ***q* = *r* JOIN *s*.**

r

A	B
a_1	b_1
a_1	b_2
a_2	b_3
a_3	b_4

s

A	C
a_1	c_1
a_1	c_2
a_3	c_3
a_3	c_4

q

A	B	C
a_1	b_1	c_1
a_1	b_2	c_1
a_1	b_1	c_2
a_1	b_2	c_2
a_3	b_4	c_3
a_3	b_4	c_4

Рис. 5. Пример естественного соединения ***q* = *r*** JOIN ***s***

9. ***θ-соединение***: пусть θ – логическая формула над атрибутов, тогда

$q\ (R \cup S) = r\ (R)$ **JOIN** $s\ (S)$ **ON** θ =

$= \{ t : (\exists t_r \in r : (\exists t_s \in s: (t_r = t[R]) \cap (t_s = t[S]) \cap \theta(t_r[R], t_s[S]))) \}$

θ-соединение аналогично естественному соединению с той лишь разницей, что условие совпадения общих атрибутов кортежей заменяется выполнением логического условия $\theta(R, S)$.

10. Операция ***переименования***

r **RENAME** $A_1, A_2, \ldots$ **AS** $B_1, B_2, \ldots$

просто изменяет наименование атрибута A_1 на B_1, A_2 на B_2 и т.д.

Реляционная алгебра – множество операций над отношениями, результатом которых является отношение. Доказано, что приведенные операции – выбора, естественного соединения, проекции, объединения, разности и переименования – образуют базис реляционной алгебры, т.е. любую операцию над отношениями можно построить при помощи операций базиса и постоянных отношений.

2.3. Реляционное исчисление

Реляционное исчисление есть множество формул вида $\{ t\ (R) : \phi(t) \}$, где t – переменная-кортеж со схемой R, $\phi(t)$ – формула над атрибутами и кортежами отношений и константами, построенная при помощи скобок, операций сравнения, логических операций, кванторов общности и существования.

Для того чтобы алгоритм вычисления отношения по формуле был конечным, множество формул ограничивают безопасными выражениями.

Например, формула $\{ t : t \notin r \}$ является опасной, так как приводит к построению множества бесконечной мощности.

Доказано, что реляционная алгебра и исчисление отношений эквивалентны по функциональной мощности: для любого выражения реляционной алгебры можно построить эквивалентную формулу исчисления отношений и наоборот.

Для реляционных БД разработаны специальные языки запросов. Среди них наиболее распространенным является Structured Query Language (SQL) – структурированный язык запросов.

Вопросы

1. Дайте определения отношения, кортежа, домена, порядка, мощности, атрибута, ключа отношения.
2. Перечислите и определите теоретико-множественные операции реляционной алгебры. Какие из этих операций трудно реализовать и почему?
3. Определите и приведите примеры операций фильтрации, проекции, естественного соединения, θ-соединения, переименования.
4. Что такое реляционная алгебра и реляционное исчисление?
5. Приведите пример запроса к данным из нескольких таблиц и запишите его в реляционной алгебре и в реляционном исчислении.

3. Проектирование реляционной БД

3.1. Требования к схеме БД

Проектирование реляционной БД заключается в построении схемы БД и определении всех ограничений, которым должны удовлетворять данные. Кроме этого, в проект БД включают бизнес-правила.

Схема БД должна удовлетворять следующим требованиям:

– БД должна быть целостной – набор отношений должен правильно описывать предметную область;

– БД не должна быть избыточной – одни и те же сведения не должны описываться дважды;

– количество отношений должно быть минимальным;

– первичные ключи должны быть минимальными;

– изменения БД должны быть минимальными при изменении предметной области;

– схема БД должна обеспечивать минимальное время выполнения запросов.

Стремление сохранить целостность и уменьшить избыточность привело к созданию нормальных форм отношений. Нормализация [2] – приведение отношений к нормальным формам – является необходимым этапом проектирования БД.

Избыточность вызывает перечисленные ниже проблемы при выполнении корректировок данных в базе (для примера рассмотрим одно отношение, содержащее все данные для схемы с рис. 1):

– добавление новых данных (например, нового поставщика) может приводить к неопределенным значениям некоторых полей (о поставке, товаре и получателе);

– удаление одних данных (для исключения данных о невыполненной поставке) может приводить к потере других, размещенных в удаляемом кортеже (например, о поставщике);

– обновление кортежа (адреса поставщика) приводит к несоответствию с данными из других кортежей.

Каждая нормальная форма позволяет устранить некоторые из этих проблем.

3.2. Функциональные зависимости. Ключи

Причина перечисленных выше проблем корректировки данных заключается в наличии зависимостей между атрибутами в отношении. Наиболее распространенным видом зависимости является функциональная зависимость.

Пусть дано отношение $\boldsymbol{r}$ со схемой $\boldsymbol{R}$ и два его подмножества $\boldsymbol{X, Y \subset R}$ атрибутов. Множество атрибутов $\boldsymbol{Y}$ функционально зависит от множества атрибутов $\boldsymbol{X}$ (обозначается $\boldsymbol{X \rightarrow Y}$), если в отношении $\boldsymbol{r}$ не может быть двух кортежей, в которых одному и тому же набору $\boldsymbol{X}$-значений соответствуют разные наборы $\boldsymbol{Y}$-значений, т.е. функциональная зависимость требует, чтобы в случае совпадения значений аргументов совпадали и значения функций. Данное поня-

тие функциональной зависимости соответствует табличному определению функции ***Y*** аргумента ***X.***

Функциональная зависимость определяется предметной областью и не зависит от экземпляра отношения (должна выполняться для всех экземпляров отношений). Связь многие к одному в модели «сущность-связь» определяет функциональную зависимость. Например, связь «поставщика» с «продажами» означает, что «продажа» является аргументом, а «поставщик» – функцией: по описанию конкретной продажи можно однозначно определить ровно одного «продавца» (но не наоборот).

Определение в качестве первичных ключей системных уникальных кодов также означает введение функциональных зависимостей – все атрибуты записи функционально зависят от такого ключа.

Одни и те же множества функциональных зависимостей могут быть заданы несколькими способами. Известно много правил и теорем вывода новых функциональных зависимостей. Ниже приведены некоторые из них.

Аксиомы Армстронга:

– аксиома рефлексивности $Y \subseteq X \Rightarrow X \rightarrow Y$ (тривиальные функциональные зависимости);

– аксиома пополнения $X \rightarrow Y \Rightarrow X \cup Z \rightarrow Y \cup Z$;

– аксиома транзитивности $X \rightarrow Y,\ Y \rightarrow Z \Rightarrow X \rightarrow Z$.

Теоремы-правила:

– правило объединения $X \rightarrow Y,\ X \rightarrow Z \Rightarrow X \rightarrow Y \cup Z$;

– правило псевдотранзитивности $X \rightarrow Y,\ W \cup Y \rightarrow Z \Rightarrow W \cup X \rightarrow Z$;

– правило декомпозиции $X \rightarrow Y,\ Z \subset Y \Rightarrow X \rightarrow Z$.

Все функциональные зависимости F^+, выводимые из множества ***F***, называют замыканием ***F*** или полным семейством зависимостей. Два множества функциональных зависимостей ***F*** и ***G*** эквивалентны, если $F^+ = G^+$. Для целей проектирования необходимо получить минимальное множество всех функциональных зависимостей предметной области. Таким множеством является неизбыточное множество функциональных зависимостей. Множество функциональных зависимостей неизбыточно, если оно не является эквивалентным любому своему собственному подмножеству.

Понятие функциональной зависимости позволяет определить ключ следующим образом. Множество атрибутов $K \subset R$ отношения ***r (R)*** с множеством функциональных зависимостей F^+ называют ключом отношения ***r***, если:

– ***R*** функционально зависит от $K\ (K \rightarrow R \in F^+)$;

– ***R*** не зависит функционально ни от какого собственного подмножества ***K***.

Один из ключей выбирают для ссылок на кортежи отношения и называют первичным ключом. Все остальные ключи называют возможными ключами. ***Ключевым атрибутом*** называют атрибут, входящий в состав какого-либо ключа, и ***неключевым*** называют атрибут, не входящий в состав ни одного ключа.

3.3. Декомпозиция и соединение

Первым требованием к БД является сохранение целостности – всех фактов и зависимостей, необходимых для адекватного отображения предметной области. Для этого могут быть предложены различные схемы БД – различные наборы отношений. В процессе реструктуризации БД важно выделить операции, которые позволяют преобразовывать схему БД без потери целостности. Такими операциями являются:

- декомпозиция – замена одного отношения несколькими;
- соединение – соединение нескольких отношений в одно.

Декомпозиция $\rho = \{r_1(R_1),..., r_n(R_n)\}$ отношения $r\ (\ R\ )$ должна содержать ту же информацию, что и отношение r. Для этого декомпозиция должна удовлетворять следующим условиям:

1. R_1 **UNION ... UNION** $R_n = R$.
2. $r_1, ..., r_n$ не должны содержать информацию, которой нет в r: $r_i = r[R_i]$.
3. $r_1, ..., r_n$ должны содержать информацию, которая есть в r: $r = r_1$ **JOIN** r_2 **JOIN ... JOIN** r_n.
4. Свойства 2 и 3 означают, что ρ обладает свойством ***соединения без потерь***.
5. Декомпозиция должна сохранять множество функциональных зависимостей. Проекцией $\pi_Z(F)$ множества функциональных зависимостей F на множество атрибутов Z называется множество функциональных зависимостей $X \to Y$, где $X, Y \subset Z, (X \to Y) \in F$. Декомпозиция ρ сохраняет множество функциональных зависимостей F, если $\pi_{R_1}(F) \cup \ldots \cup \pi_{R_n}(F)$ эквивалентно F.

Свойство соединения без потерь и сохранения множества функциональных зависимостей не следуют одно из другого.

3.4. Нормальные формы

Формализация процесса построения схемы реляционной БД, изучение аномалий, возникающих при корректировке БД, привели к формулировке ряда требований. Если отношение удовлетворяет требованиям, то говорят, что оно находится в соответствующей нормальной форме. Процесс приведения схем отношений к нормальной форме называют нормализацией.

Первая нормальная форма (1НФ) является самой простой.

Отношение находится в 1НФ, если каждый атрибут отношения является атомарным, т.е. неделимым в смысловом отношении.

Примерами атрибутов, нарушающих 1НФ, являются адреса, списки и вообще любые агрегаты данных.

Вторая нормальная форма (2НФ) связана с исключением из отношения неполных (частичных) функциональных зависимостей неключевых атрибутов от ключа. ***Неполная*** или ***частичная*** функциональная зависимость в отношении r атрибута A от ключа K существует, если существует собственное подмножество Y ключа K и функциональная зависимость A от Y. Если такого подмножества у нет, то зависимость A от ключа K называется ***функционально полной***.

Отношение находится в 2НФ, если оно находится в 1НФ и каждый неключевой атрибут функционально полно зависит от ключа.

В отношении с неполными функциональными зависимостями возникает избыточность, что приводит к проблемам сохранения целостности при добавлении записей и к возможной потере информации о неполной зависимости при удалении. Например, пусть итоги сессии хранятся в отношении «Оценки» с атрибутами «Номер студ.билета», «Фамилия студента», «Номер группы», «Предмет», «Преподаватель» и «Оценка». Пусть существуют следующие зависимости:

– «Номер студ.билета» → «Фамилия студента»;
– «Номер студ.билета» → «Номер группы»;
– «Номер студ.билета», «Предмет» → «Оценка»;
– «Номер группы», «Предмет» → «Преподаватель».

Ключом отношения является комбинация атрибутов «Номер студ.билета» и «Предмет». Зависимости «Номер студ.билета» → «Фамилия студента», «Номер студ.билета» → «Номер группы» являются неполными. Для приведения данного отношения в 2НФ необходимо вывести из него указанные зависимости, например выполнить декомпозицию на отношения

(«Номер студ.билета», «Фамилия студента», «Номер группы») и
(«Номер студ.билета», «Предмет», «Преподаватель», «Оценка»).

Третья нормальная форма (3НФ) исключает не только неполные, но и транзитивные зависимости. ***Транзитивная зависимость*** $K \to A$ атрибута A от ключа K существует, если существует множество атрибутов $Y \subset R$ такое, что $Y \to A$.

Отношение r(R) находится в 3НФ, если оно находится в 2НФ и каждый неключевой атрибут A нетранзитивно зависит от ключа X.

Транзитивные зависимости также приводят к избыточности и проблемам обновления. Поэтому транзитивные зависимости от ключа необходимо описывать отдельным отношением. Например, в отношении («Номер студ.билета», «Номер комнаты», «Телефон»), описывающем распределение студентов по комнатам общежития, есть зависимости «Номер студ.билета» → «Номер комнаты», «Номер комнаты» → «Телефон». Атрибут «Телефон» транзитивно зависит от ключа «Номер студ.билета».

Нормальная форма Бойса – Кодда (НФБК) применяется, если в отношении r ключевой атрибут транзитивно зависит от ключа. Определение НФБК использует понятие детерминанта. Множество атрибутов X является ***детерминантом*** B, если $X \to B$ и ни одно собственное подмножество X не обладает этим свойством.

Отношение r (R) находится в НФБК, если оно находится в 2НФ и каждый детерминант из R является возможным ключом.

Например, пусть значение кортежа (a, f, o) заключается в том, что фирма f покупает билеты в аэропорту a и оплачивает билеты отделение фирмы o. Отношение содержит две зависимости $(a, f) \to o$, $o \to f$. Ключами отношения явля-

ются ***{a, o}*** и ***{a, f}***. Фирма ***f*** является ключевым атрибутом, но зависит от атрибута ***o***, который является детерминантом, но не является ключом отношения. Зависимость ***o→f*** приводит к избыточности и должна быть выведена из отношения.

Нормальная форма Бойса – Кодда обеспечивает отсутствие аномалий корректировки, добавления и удаления данных, если все зависимости являются функциональными. Однако кроме функциональных существуют зависимости других видов, в частности многозначная зависимость. Например, в отношении «Расписание» на рис. 6 нет функциональных зависимостей и оно находится в третьей нормальной форме, тем не менее его можно представить как результат соединения двух более простых отношений «Дни вылетов» и «Самолеты рейсов».

Это возможно благодаря существованию многозначных зависимостей «Рейс»→→ «День недели» и «Рейс»→→ «Тип самолета». Объединение нескольких многозначных зависимостей приводит к появлению избыточности. Многозначная зависимость наступает тогда, когда значениями функций являются множества. Более точным является следующее определение.

«Расписание»

Рейс	День недели	Тип самолета
106	пн	747
106	чт	747
106	пн	1011
106	чт	1011
204	ср	707
204	ср	727

«Дни вылетов»

Рейс	День недели
106	пн
106	чт
204	ср

«Самолеты рейсов»

Рейс	Тип самолета
106	747
106	1011
204	707
204	727

Рис. 6. Пример многозначной зависимости и ее декомпозиции

В отношении ***r(A, B, C)*** **существует многозначная зависимость** ***A→→B,*** если множество значений ***{b∈B},*** соответствующих паре (***a, c), a∈A, c∈C,*** зависит только от ***a*** и не зависит от ***c***.

Из определения следует, что при наличии в отношении ***r*** кортежей (a_1, b_1, c_1) и (a_1, b_2, c_2) в нем (по причине многозначной зависимости) должны быть кортежи (a_1, b_2, c_1) и (a_1, b_1, c_2), в этом случае в отношении кроме зависимости $A \to\to B$ присутствует зависимость $A \to\to C$. Приведенное в примере отношение находится в 3НФ (ключ ABC), но в то же время разлагается без потерь на отношения со схемами AB и AC.

Отношение находится в четвертой нормальной форме (4НФ), если в случае существования многозначной зависимости A→→B все остальные атрибуты функционально зависят от AB.

Существует еще и пятая нормальная форма, связанная с зависимостью соединения. Зависимость соединения обобщает многозначную зависимость на

случай, когда декомпозиция без потерь возможна не на два, а на большее число отношений (рис. 7).

В целом построение реляционной модели данных можно представить следующим образом:

1. Получение некоторого начального набора отношений на основе модели «сущность-связь».

2. Определение всех функциональных, многозначных и другого вида зависимостей.

3. Построение минимального покрытия – неизбыточного множества функциональных зависимостей.

4. На основе минимального покрытия выполнение нормализации отношений при помощи операций декомпозиции и соединения.

5. Оценка полученной схемы БД с точки зрения безопасности, секретности и эффективности использования и хранения данных. При необходимости предлагается иное деление БД на отношения и пункт 4 выполняется заново.

r

A	B	C
a1	b1	c1
a1	b2	c2
a2	b3	c3
a3	b3	c4
a4	b4	c5
a5	b5	c5

r_1

A	B
a1	b1
a1	b2
a2	b3
a3	b3
a4	b4
a5	b5

r_2

A	C
a1	c1
a1	c2
a2	c3
a3	c4
a4	c5
a5	c5

r_3

B	C
b1	c1
b2	c2
b3	c3
b3	c4
b4	c5
b5	c5

Рис. 7. Пример зависимости соединения

3.5. Ограничения реляционных языков и некоторые ошибки проектирования

При проектировании БД необходимо учитывать специфику реляционных языков, в основе которых лежит реляционная алгебра. Первый пример [6] связан с отсутствием средств построения списков колонок с определенными свойствами. В таблице с рис. 8 каждому элементу соответствует одно поле. Запросы о составе вещества (элементы, составляющие не менее 1 % вещества) должны возвращать список атрибутов записи, удовлетворяющих условию, а такая операция не предусмотрена реляционной алгеброй. Списками в реляционных БД являются строки, а не столбцы. Поэтому правильнее разбить это отношение на три различных отношения:

– ВЕЩЕСТВА(НОМ_ВЕЩЕСТВА, ВЕЩЕСТВО);

– ЭЛЕМЕНТЫ(НОМ_ЭЛЕМЕНТА, ЭЛЕМЕНТ);

– ХИМИЧЕСКИЙ_СОСТАВ_ВЕЩЕСТВ(НОМ_ВЕЩЕСТВА, НОМ_ЭЛЕМЕНТА, ПРОЦЕНТ).

Таблицы, аналогичные таблице с рис. 8 (их еще называют сводными или шахматными), довольно часто встречаются в качестве итоговых. Например, продавцы в качестве заголовков строк, потребители в качестве заголовков колонок, а каждая клетка содержит стоимость приобретенных товаров. Такие

данные в реляционной модели удобнее (с точки зрения построения запросов) хранить в отношении с атрибутами «продавец», «покупатель», «стоимость».

Химический состав веществ

Наименование вещества	Водород	Гелий	…	105 элемент
Дезоксирибонуклеиновая кислота	5	3	…	0.01
Бензин	50	0	…	0
…	…	…	…	…

Рис. 8. Описание состава веществ по элементам

Вторая достаточно распространенная ошибка – это разбиение множества объектов на классы и описание каждого класса отдельным отношением. Например, таблицу на рис. 8 можно разбить на 105 отношений:

– ВЕЩЕСТВА_СОДЕРЖАЩИЕ_ВОДОРОД(НОМ_ВЕЩЕСТВА, ПРОЦЕНТ);

– ВЕЩЕСТВА_СОДЕРЖАЩИЕ_ГЕЛИЙ(НОМ_ВЕЩЕСТВА, ПРОЦЕНТ);

– …

– ВЕЩЕСТВА_СОДЕРЖАЩИЕ_ЭЛЕМЕНТ105(НОМ_ВЕЩЕСТВА, ПРОЦЕНТ).

В этом случае для ряда запросов придется формировать список не атрибутов, а отношений. Такие возможности также не предусмотрены реляционной алгеброй и создают алгоритмические проблемы. Даже простое разбиение таблицы товаров на две – проданные и не проданные товары – требует неоправданного усложнения запросов.

Вопросы

1. Перечислите требования к схеме БД. Какие из этих требований являются более важными и почему?

2. Дайте определение понятий «функциональная зависимость», «ключ», «ключевой атрибут», «неключевой атрибут», «неизбыточное множество функциональных зависимостей», «функционально полная зависимость», «функционально неполная зависимость», «частичная зависимость», «транзитивная зависимость», «детерминант», «многозначная зависимость».

3. Когда два множества функциональных зависимостей считаются эквивалентными?

4. Может ли атрибут быть ключевым в одном отношении и неключевым в другом? Приведите примеры.

5. В чем смысл декомпозиции и соединения?

6. Как выполнить декомпозицию без потери информации?

7. Дайте определение каждой нормальной формы, укажите, какие проблемы изменения данных связаны с нарушением каждой из них.

8. Перечислите шаги построения реляционной модели.

4. Структурированный язык запросов SQL

Прообраз языка SQL возник в 1970 г. в рамках научно-исследовательского проекта System/R, работа над которым велась в лаборатории Санта-Тереза фирмы IBM. Сегодня SQL – это фактический стандарт интерфейса с современными реляционными СУБД. Язык SQL имеет официальный стандарт – ANSI/ISO. Большинство разработчиков СУБД придерживаются этого стандарта, однако часто расширяют его для реализации новых возможностей обработки данных. Новые механизмы управления данными могут быть использованы только через специальные операторы SQL, в общем случае не включенные в стандарт языка.

SQL не является языком программирования в традиционном представлении. На нем пишутся не программы, а запросы к базе данных. В отношении запросов SQL является декларативным языком. Это означает, что с его помощью можно сформулировать, что необходимо получить, но нельзя указать, в какой последовательности это следует сделать.

В качестве иллюстраций рассматривается построение и выполнение запросов с использованием СУБД MS SQL server. Для записи и передачи запросов серверу в этой СУБД используется программа MS SQL server Management Studio.

4.1. Применение MS SQL server Management Studio для работы с запросами

После запуска Management Studio открывается окно подключения к серверу (Рис. 9). Имя сервера обычно является именем компьютера в сети. Прочие параметры устанавливаются по умолчанию и не нуждаются в изменении.

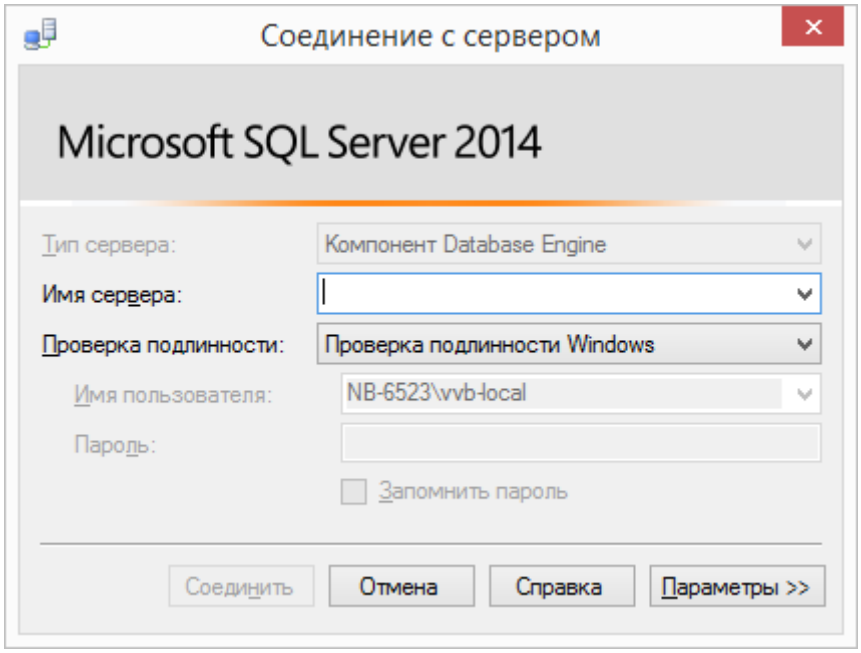


Рис. 9. Окно подключения к серверу

Команды в Management Studio можно задавать с помощью меню, панели инструментов или с помощью контекстного меню. В основном будет использоваться последний вариант, потому что его применение позволяет видеть коман-

ды, относящиеся к указанному курсором объекту. На Рис. 10 для базы данных «Торговля» выбор команды «Создать запрос» в контекстном пеню приводит к созданию окна запросов для данной БД. Можно одновременно открыть несколько окон. Кнопка «! Выполнить» приводит к передаче команды серверу. После выполнения команды результат, возвращенный сервером, будет отображен в нижней части окна.

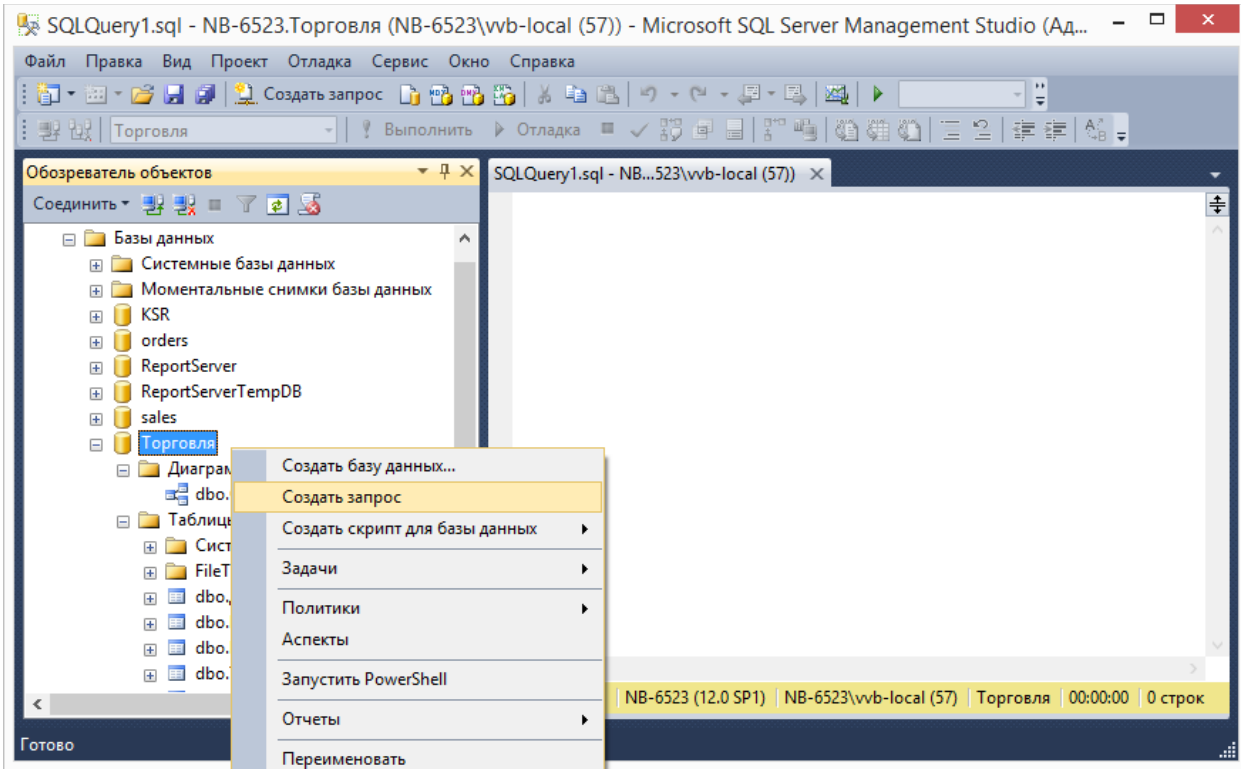


Рис. 10. Создание окна запросов

4.2. Типы данных

Язык SQL / 89 поддерживает все базовые типы данных:

– BLOB – Binary large object произвольной величины для хранения графики, текстов;

– CHAR(n) – строки ровно из n знаков: строки длиной меньше n дополняются пробелами;

– VARCHAR (n) – строки не более n знаков (n может быть от 1 до 32 767), хранится реально указанное число знаков;

– DATETIME – длинная дата (8 байт) для дат из интервала 1 января 100 – 11 января 5941;

– SMALLDATETIME – короткая дата (4 байта) от 1.01.1900 до 6.6.2079;

– DECIMAL (p, s), NUMERIC (p, s) – числа с указанным общим количеством цифр p и количеством дробных знаков s;

– REAL – вещественное число (4 байта);

– FLOAT – вещественное число двойной точности (8 байт);

– INTEGER – целое (4 байта) от –2 147 483 648 до +2 147 483 648;

– SMALLINT – целое (2 байта) от –32 768 до +32 767.

MS SQL server сервер поддерживает перечисленные типы данных, а также дополняет эти типы многими другими. В частности, рекомендуется использовать тип NVARCHAR (n) – строки переменной длинны в кодировке UNICOD. Применение этой кодировки обеспечивает независимость представления строк в БД от кодовой страницы, установленной в информационной системе.

Использование числовых форматов в SQL не отличается от их применения в других языках программирования. Результат операции с целыми числами снова преобразуется к целому числу. В случае деления это будет приводить к отбрасыванию дробной части числа, так что 1/3 будет равно 0. Для вещественных чисел может появится эффект перевода из десятичной системы счисления в двоичную и наоборот. Вполне возможно получение значений 0.999999999 и 0.10000000001.

Даты на самом деле представлены вещественными числами. Целая часть числа – это количество целых дней с начальной даты, а дробная часть – это время. Число 43861,25 соответствует шести утра 2 февраля 2020 г.

4.3. Создание и изменение структуры таблицы. Табличные ограничения

Команда создания таблицы кроме имени таблицы определяет список колонок и ограничений:

CREATE TABLE <имя таблицы> (<колонка> [, <колонка>…] [, <табличное ограничение>[,<табличное ограничение> …]])

Пункт <колонка> задает описание одной колонки (поля) таблицы:

<колонка> ::= <имя колонки > <тип данных> [<свойства и ограничение колонки>]

Типом данных может быть любой из перечисленных в предыдущем параграфе.

Свойства и ограничения могут быть следующими:

– DEFAULT задает значения по умолчанию, например

COUNTRY VARCHAR(20) default ‘Российская Федерация’;

– IDENTITY(n,s) определяет автоинкрементное поле, значения которого формируются автоматически с начальным значением n и увеличиваются при добавлении записи на s;

– свойство NULL (NOT NULL) разрешает (запрещает) задавать пустые значения поля;

– ограничение колонки *CHECK (<условие>)* задает условие, образованное при помощи операций: конъюнкции (and), дизъюнкции (or) и отрицания (not), скобок и сравнений вида *<выражение> <знак сравнения> <выражение>*.

Кроме этого, возможно употребление условий, использующих команду выбора *SELECT* (см. описание команды ниже).

Значение NULL является специальной меткой и не совпадает с пустой строкой или нулем. Пустые значения могут породить проблемы при вычислении, когда один из операндов равен NULL. Результат выражения с NULL-значением может быть снова NULL, или NULL-значение интерпретируется как ноль в зависимости от параметров сервера. Другая ситуация возникает при сравнении двух NULL-значений: NULL-значения считаются не равными друг другу.

Программы администрирования БД предоставляют современный графический интерфейс для создания и корректировки структуры таблицы. В Management Studio для создания таблицы используется команда «Создать таблицу» («New table»), для корректировки структуры таблицы – команда «Проект» («Design»). Выполнение этих команд приводит к созданию диалогового окна (Рис. 11), в котором можно задать все свойства поля. Команда «сохранить» приведет к передаче серверу соответствующей команды «Create table …» в случае создания новой таблицы или «Alter table» в случае изменения структуры существующей таблицы.

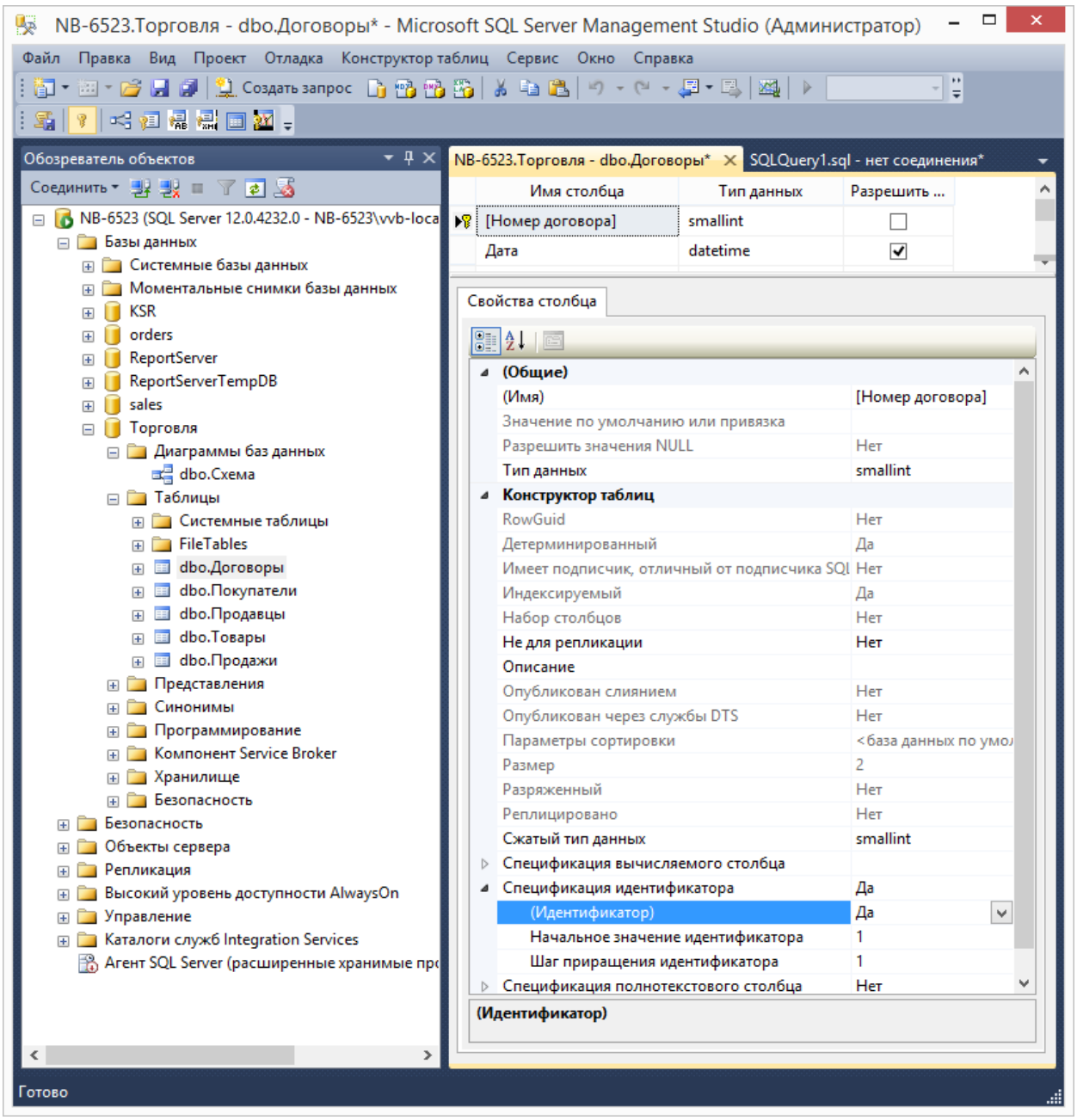


Рис. 11. Окно создания/изменения структуры таблицы

Табличные ограничения устанавливают свойства таблицы в целом, такие как уникальность значений, или устанавливают соотношения данных таблицы с

данными других таблиц. В SQL табличное ограничение имеет следующее описание: *CONSTRAINT <имя ограничения> <ограничение>* и может быть следующим:

1. Ограничение «первичный ключ» *PRIMARY KEY (<список имен колонок через запятую>)* указывает, что данная последовательность полей является первичным ключом, т.е. не допускается существования двух записей с одинаковыми значениями первичного ключа. В базе данных создает индекс для повышения скорости поиска по первичному ключу. Поля, составляющие первичный ключ, не могут принимать пустые значения (обязательно должны быть NOT NULL).

2. Ограничение «внешний ключ» *FOREING KEY (<имена колонок таблицы через запятую>) REFERENCES <имя справочной таблицы> (<имена колонок справочной таблицы через запятую>)* указывает, что значения перечисленных полей обязательно должны быть в первичном ключе некоторой записи справочной таблицы. Таким образом, в справочной таблице должен быть определен первичный ключ по указанным полям. Первичный и внешний ключи являются основными инструментами определения ссылочной целостности. В справочной таблице первичный ключ указывает ровно одно описание объекта, в другой таблице внешний ключ задает ссылку на объект. Ссылочная целостность заключается в корректности каждой ссылки. В таблице нельзя определить запись со ссылкой на несуществующую запись в справочной таблице (нельзя указать несуществующее значение первичного ключа), а в справочной таблице нельзя удалить запись, если на значение первичного ключа записи есть ссылки. Например, в таблице «Договоры» определен внешний ключ

CONSTRAINT [FK_Договоры_Продавцы] FOREIGN KEY
([Код Продавца]) REFERENCES [Продавцы] ([Код продавца]).

В этом случае в этой таблице невозможно указать «Код продавца», которого нет в таблице «Продавцы», а в таблице «Продавцы» – удалить запись или изменить поле «Код продавца», если на соответствующую запись есть ссылки.

3. Ограничение *CHECK(<условие>)* позволяет определить ограничение в виде произвольного логического выражения. Например, построить запрос на определение достаточного количества денежных средств на счете покупателя.

Management Studio предоставляет графический интерфейс для определения ограничений ссылочной целостности: первичных и внешних ключей. Для каждой базы данных можно определять диаграммы для описания множеств связанных таблиц. Для создания или изменения диаграммы можно в обозревателе объектов сервера для группы объектов «Диаграммы базы данных» выбрать в контекстном меню команду «Создать…» или «Изменить». В результате откроется окно (Рис. 12), в котором будут отображены таблицы со списком полей, первичные ключи (изображение ключа рядом с ключевым полем) и внешние ключи в виде линий, соединяющих таблицы. На диаграмме можно добавлять новые таблицы, удалять и добавлять первичные ключи, удалять внешние ключи с помощью соответствующих команд контекстного меню, привязанного к объекту. Например, для создания или удаления первичного ключа можно указать поле и в контекстном меню (нижняя стрелка на Рис. 12) выбрать команду «Со-

здать…» или «Удалить…». Для создания внешнего ключа достаточно перетащить поле из одной таблицы на соответствующее поле в другой (верхняя стрелка на Рис. 12).

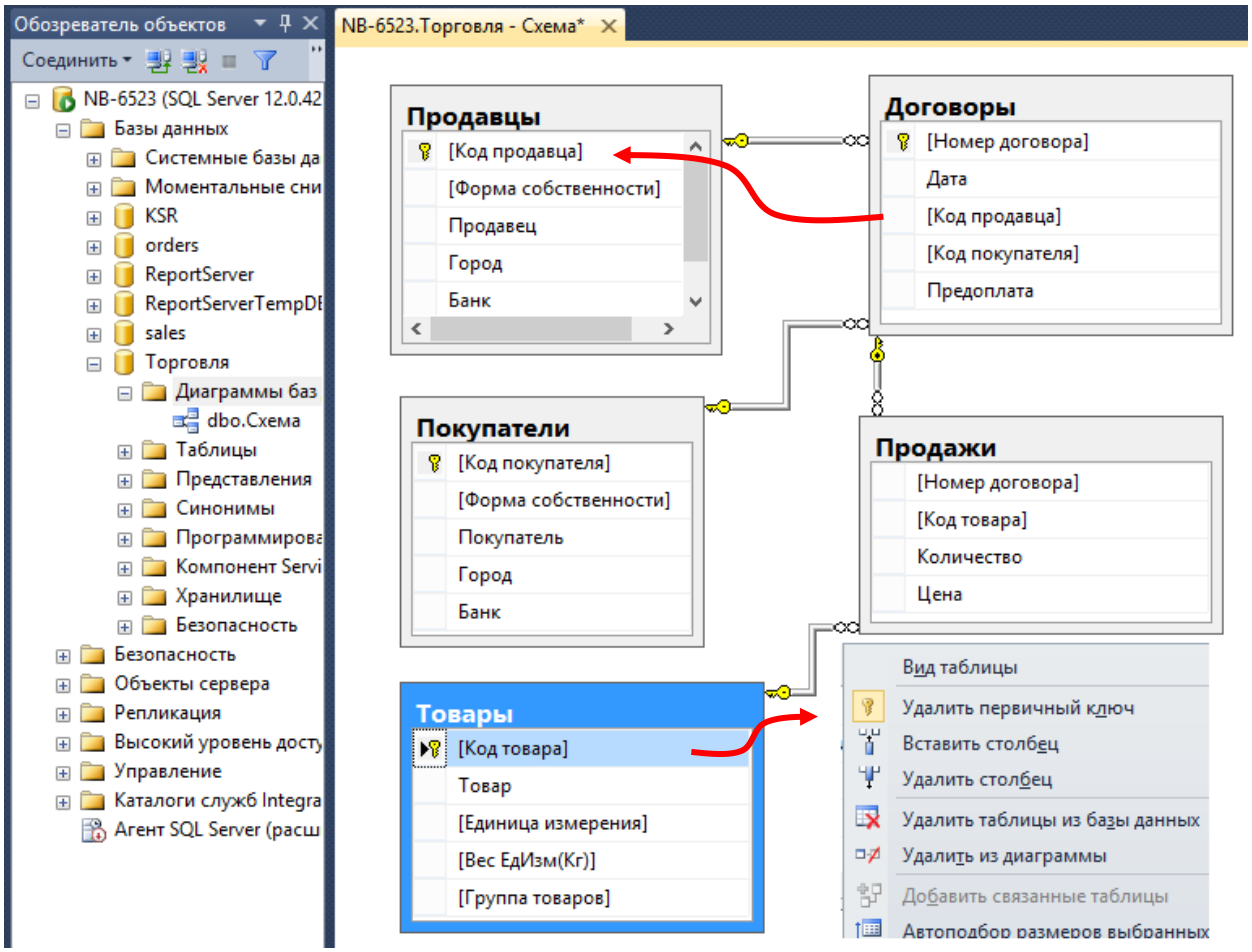


Рис. 12. Диаграмма базы данных

Все эти действия приведут к созданию соответствующих SQL-команд и передаче их серверу в момент сохранения диаграммы.

Команда изменения структуры таблицы имеет следующий вид:

ALTER TABLE <таблица> <изменения через запятую>

Изменение описывается фразой

ADD <колонка> |

ADD <Табличное ограничение> |

DROP <колонка>l

DROP CONSTRAINT <ограничение>

Команду создания объекта можно получить с помощью команды контекстного меню «Создать скрипт для…» \ «Используя Create». Ниже приведены команды создания таблицы «Договоры»:

CREATE TABLE [Договоры] (

[Дата] [smalldatetime] NOT NULL ,

[Номер договора] [int] NOT NULL ,

[Код Продавца] [int] NOT NULL ,

[Предоплата] [float] NOT NULL)

ALTER TABLE [Договоры] ADD

CONSTRAINT [PK_Договоры] PRIMARY KEY CLUSTERED ([Номер договора])

ALTER TABLE [Договоры] ADD

CONSTRAINT [FK_Договоры_Продавцы] FOREIGN KEY

([Код Продавца]) REFERENCES [Продавцы] ([Код продавца])

Команда удаления таблицы *DROP TABLE <имя таблицы>* удаляет таблицу со всеми записями и ограничениями.

Для таблицы можно определить несколько индексов. При определении первичного ключа индекс строится автоматически. Индекс является эффективным средством поиска записей в таблице по значению. Основой организации индекса обычно является конструкция B-дерева. B-дерево состоит из страниц. Каждая страница содержит некоторое ключей и указателей на страницы-потомки, причем 0-й потомок содержит ключи меньшие 1-го ключа, k-й потомок содержит ключи из интервала (k-й ключ, (k + 1)-й ключ), последний потомок содержит ключи большие последнего ключа (рис. 13). Таким образом, количество указателей (потомков страницы) на 1 больше количества ключей.

Страница может быть незаполненной. Если все страницы заполнены полностью, то при добавлении значения ключа приходится выполнять добавление страниц и реорганизацию дерева. Неполные страницы позволяют уменьшить количество реорганизаций дерева.

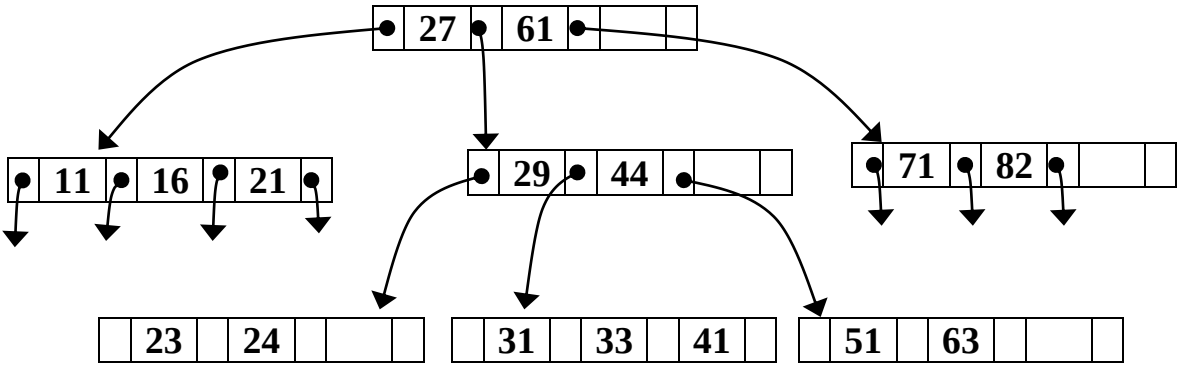


Рис. 13. Фрагмент B-дерева

Поиск ключа начинается с корневой страницы. Если ключ на странице не найден, то однозначно определяется станица-потомок, на которой поиск следует продолжить. Максимальное количество просматриваемых страниц равно высоте дерева. С увеличением количества сложность поиска растет как логарифм количества ключей, причем основанием логарифма является среднее количество ключей на странице. Например, при среднем количестве ключей на странице 10 для поиска ключа среди миллиона достаточно просмотреть всего lg(1 000 000)=6 страниц, а при увеличении количества ключей в 10 раз придется просматривать всего на одну страницу больше.

Один из индексов может быть кластерным – на терминальных страницах ключи хранятся вместе с записью. Кластерный индекс не требует дополнительных расходов памяти, но для таблицы можно определить только один кластерный индекс. Некластерные индексы требуют память для хранения.

Индекс создается командой

CREATE [UNIQUE] [CLUSTERED | NONCLUSTERED] INDEX <индекс>
ON <таблица> (<поля>)
[WITH < опции > [,...n]]
[ON <группа_файлов>]

Параметры команды означают следующее:

– *UNIQUE* – обеспечивает создание уникального индекса;

– *CLUSTERED* – записи хранятся в индексном порядке, страницы-листья содержат данные. У таблицы может быть только один кластерный индекс;

– *NONCLUSTERED* – индекс хранится в отдельных страницах. Последовательность записей не совпадает с индексной;

– *<индекс>* – уникальное имя в рамках БД для указания индекса;

– *<поля>* – список полей через запятую. После каждого поля можно указывать направление сортировки *ASC* – по возрастанию или *DESC* – по убыванию. Индексный порядок записей соответствует их сортировке сначала по первому полю, затем по второму и т.д.;

– *< опции >* могут быть следующими:

- *PAD_INDEX* – резервирует свободное пространство для индексных записей в момент создания, употребляется вместе с параметром *FILLFACTOR;*
- *FILLFACTOR = <процент заполнения>* – указывает долю заполнения пространства индексных записей на каждой странице. Используется при создании таблицы. Если эта доля равна 100 %, то ключами заполняется все пространство. Это экономит память, однако, при добавлении или удалении записей потребуется перекомпоновка страниц на пути от листовой страницы до корневой. Если доля меньше 100 %, то такая перекомпоновка требуется реже;
- *IGNORE_DUP_KEY* – игнорирует появление дублирующих значений ключа;
- *DROP_EXISTING* – новый ключ заменяет одноименный существующий;
- *STATISTICS_NORECOMPUTE* – отменяет рекомпиляцию статистики, используемой для оптимизации выполнения запроса;
- *SORT_IN_TEMPDB* – указывает использование БД временных данных для построения индекса. Это может повысить скорость создания индекса, если эта БД расположена о другом диске или в оперативной памяти;
- *ON <группа_файлов>* – физически размещает индекс в указанной группе.

Индекс также можно создать при помощи диалоговых команд в программе Management Studio.

При правильном построении и использовании индексы обеспечивают более высокую производительность выполнения запросов.

4.4. Команды изменения содержания таблицы

Команда добавления записей в таблицу

INSERT INTO <таблица> [(<колонка> [, <колонка> ...])]

{VALUES (<значение> [, <значениеl> ...]) | <select – выражение>}

состоит из двух частей: список полей и список значений. Команда может не иметь списка полей добавляемых записей. В этом случае должны быть заданы значения всех полей. Первое значение записывается в первое поле, второе – во второе и т.д. Типы полей и значений должны совпадать или допускать корректное преобразование. Например,

INSERT INTO [Продавцы]

([Код продавца],[Форма собственности],[Продавец],[Город],[Банк])

VALUES (12, 'АОЗТ', 'Прогресс', 'Ангарск', 'Надежный')',

В описанном варианте команда *INSERT* добавляет одну запись. Поля, отсутствующие в списке полей команды, получают значение NULL.

Существует еще один вариант команды *INSERT*, в нем добавляемые записи также могут быть определены командой выбора *SELECT*. В этом случае могут быль добавлены несколько записей.

Команда *DELETE FROM <таблица> [WHERE <условие>]* удаляет из указанной таблицы все записи, удовлетворяющие условию. Например, команда *DELETE FROM [Продавцы] WHERE [Код продавца]=12* удалит запись о продавце с кодом 12. Если условие не указано, то выполняется попытка удалить все записи.

Команда модификации записей

UPDATE <Таблица> SET <колонка> = <выражение>

[,<колонка> = < выражение >...]

[WHERE <условие>]

заменяет значения полей указанными выражениями для всех записей, удовлетворяющих условию. Например,

UPDATE [Продавцы] SET [Банк] = 'Универсал'

WHERE [Банк] = 'ВСКБ'

переводит продавцов-клиентов банка «ВСКБ» в банк «Универсал».

В Management Studio для каждой таблицы можно командой «Изменить…» контекстного меню открыть табличную форму изменения таблицы (Рис. 14). Изменение строки в этой форме передаст серверу команду UPDATE. Для добавления строк предусмотрена последняя пустая строка, заполнение ее и сохранение данных приведут к генерации соответствующей команды INSERT. В этой форме можно выделить строку и запустить команду ее удаления – на сервер будет передана команда DELETE. По аналогичному сценарию работают все программы, корректирующие данные в БД. При этом пользователи могут не видеть часть полей. Например, программы корректировки данных вместо кодов продавцов демонстрируют их наименования. Однако какими бы сложными не были сценарии корректировки данных, они приведут к отправке серверу команд UPDATE, INSERT и DELETE.

На практике самой редкой командой является удаление записей. Это связано с тем, что практически все данные, записанные в БД, могут понадобиться для обработки или поиска. Нет необходимости опасаться роста объема данных. Во-первых, гораздо хуже, когда необходимых данных нет в базе, во-вторых, рост объема данных практически не приводит к сколько-нибудь значимому снижению производительности сервера, в-третьих, возможности вычислительной техники растут значительно быстрее объемов данных в базе.

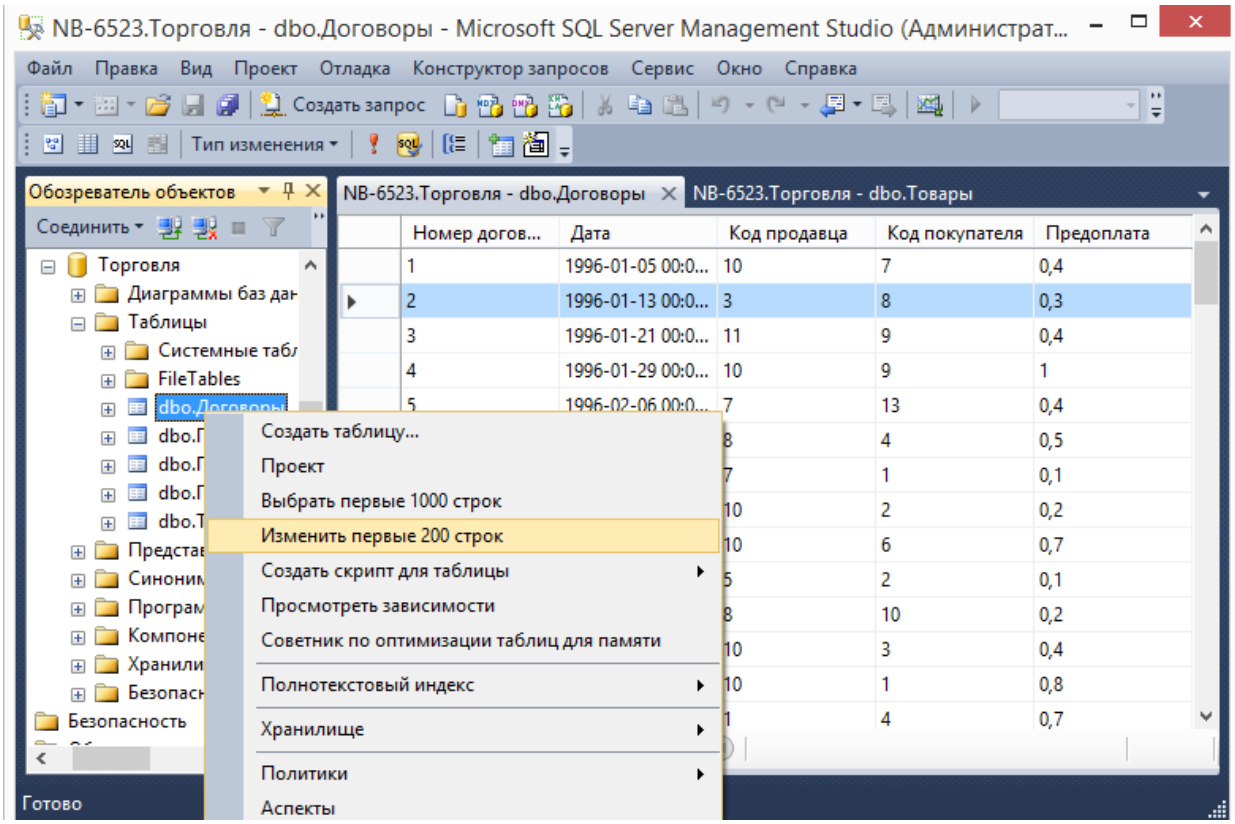


Рис. 14. Команда и форма корректировки записей таблицы

4.5. Выбор данных

Базы данных используют для накопления и обработки данных. Традиционно применение БД связывают с технологией OLTP (Online Transaction Processing), в которой под транзакцией понимаются команды корректировки или выбора данных. Базовой командой выбора является команда SELECT. Как обычно, результат запроса на выбор данных в БД оформляется в виде базовой структуры данных БД, для реляционных БД это таблица. Таким образом, SELECT выбирает данные из нескольких таблиц и возвращает результат в виде таблицы. Даже если это будет одно значение, результатом будет таблица из одного столбца и одной строки. Запрос может вернуть таблицу, в которой нет записей. Это не будет ошибкой, так бывает, если в БД нет данных, удовлетворяющих условиям запроса.

Команда SELECT имеет следующий формат:

SELECT [DISTINCT] <список выражений>

[INTO <таблица-результат>]
FROM <соединение таблиц>
[WHERE <условие>]
[Group BY <список выражений группировки>
[HAVING <условие выбора группы>]]
[ORDER BY <список полей сортировки>]

Квадратные скобки говорят о том, что соответствующие части команды могут отсутствовать, однако нельзя изменять порядок их следования.

Список выражений определяет набор колонок в результирующей таблице. Фраза *INTO* сохраняет результат в БД, это используется достаточно редко, чаще всего результат выбора передается программе-клиенту и запоминать выбранные данные не требуется. Пункт *FROM* определяет, из каких таблиц будут выбраны данные. Условие *WHERE* с помощью логических выражений определяет, какие данные будут выбраны. Фраза *Group BY* позволяет разбить исходные данные на группы и для каждой группы сформировать запись со сводными данными. Условие *HAVING* позволяет задать логическое условие выбора, использующее сводные данные по группе. Пункт *ORDER BY* определяет порядок сортировки записей в результирующей записи.

4.5.1. Соединение таблиц в запросе

В пункте *FROM* определяются соединения таблиц, из которых извлекается информация. Для этого может использоваться оператор соединения

<табличный источник> <оператор соединения> <табличный источник> ON <условия соединения>

Табличный источник состоит из имени таблицы, за которым можно через пробел указать локальный псевдоним, заменяющий имя таблицы только внутри команды SELECT. Если псевдоним не указан, то он полагается равным имени таблицы. Псевдонимы позволяют использовать несколько экземпляров одной и той же таблицы в одном запросе. Вместо таблицы может быть другая команда SELECT, заключенная в круглые скобки. В этом случае псевдоним должен быть обязательно. Условие соединения обычно имеет вид равенств

<псевдоним таблицы1>.<поле1>=<псевдоним таблицы2>.<поле2>

Для формулировки условия используются логические операторы AND, OR, NOT.

Условия соединения могут быть указаны и в пункте WHERE. В этом случает в пункте FROM таблицы просто перечисляются через запятую. Соединение в SELECT аналогично оператору соединения в реляционной алгебре, т.е. каждая запись первой таблицы комбинируется с теми записями второй таблицы, которые удовлетворяют условию соединения. Каждая такая комбинация является основой для построения записи результирующей таблицы. Например, соединение

Договоры O INNER JOIN Продавцы S
ON O.[Код Продавца] = S.[Код продавца]

к каждой записи о договоре добавляет запись о соответствующем продавце.

Оператор соединения имеет следующие варианты:

– *[INNER] JOIN* – обычное (внутреннее) соединение в описанном выше смысле;

– *[LEFT | RIGHT | FULL] [OUTER] JOIN* – левое | правое | полное внешнее соединение – в результат включаются даже те записи левой | правой | обеих таблиц, которым не соответствую записи другой таблицы;

– *CROSS JOIN* – декартово произведение таблиц: формируются всевозможные комбинации записей из левой и правой таблицы. Эта операция достаточно опасная – может привести к созданию большого количества записей, для хранения которых может не хватить ресурсов.

Для указания последовательности соединения можно использовать скобки.

4.5.2. Определение полей таблицы, формируемой запросом

Список выражений, приведенных после ключевого слова *SELECT*, определяет состав и последовательность колонок в результирующей таблице. Ключевое слово *DISTINCT* означает, что в этой таблице все записи будут уникальны – дубликаты записей будут вычеркнуты. Выражения (в простейшем случае поля) перечисляются через запятую. Выражение определяется фразой *<выражение> [AS <имя поля>]*, в которой после слова «AS» указывается имя поля в итоговой таблице (можно не указывать, если выражение является просто полем).

Выражение может быть представлено:

– *полем* базы данных в формате [<имя | псевдоним таблицы>.]<поле>. Имя или псевдоним таблицы необходимо указывать, если поле входит в состав нескольких соединяемых таблиц;

– *константой* (указанное значение константы появится в каждой записи выходной таблицы);

– *выражением*, построенным при помощи констант, полей, функций, знаков операций и скобок;

– *статистической функцией*, вычисляемой по группе записей:

- *AVG(<выражение>)* – среднее значение выражения;
- *COUNT(*)* – число записей, *COUNT(<поле>)* – число непустых значений поля, *COUNT(DISTINCT <поле>)* – число различных значений поля;
- *MIN(<выражение>)* – наименьшее значение выражения;
- *MAX(<выражение>)* – наибольшее значение выражения;
- *SUM(<выражение>)* – сумма выражений.

Статистическая функция вычисляется либо по группе записей исходной таблицы в случае группировки данных, либо по всей исходной таблице в запросе без группировки.

Выражение может быть арифметическим – для его построения используют операторы + , –, *, /, % (остаток от деления), скобки и функции. Набор арифметических функций является обычным для языков программирования, например:

– *CEILING(<число>)* – возвращает ближайшее большее целое число;

– *FLOOR(<число>)* – возвращает ближайшее меньшее целое число;

– *RAND([<число>])* – возвращает псевдослучайное равномерно распределенное в интервале [0,1) число, следующее за указанным;

– *ROUND (<число>, <число знаков дробной части> [, <числовой тип>])* – возвращает число, округленное до указанного числа знаков.

Для работы с датами предусмотрены следующие функции:

– *DATEADD (<единица измерения>, <число>, <дата>)* – возвращает дату увеличенную на заданное число единиц. Единица может быть годом (y), кварталом (q), месяцем (m), днем (d) или более детальной;

– *DATEDIFF (<единица измерения>, <начальная дата>, <конечная дата>)* – возвращает количество единиц времени прошедших с начальной до конечной даты;

– *DATEPART (<единица измерения>, <дата>)* – возвращает указанную часть даты;

– *GETDATE ( )* – возвращает дату сервера;

– *YEAR(<дата>), MONTH(<дата>), DAY(<дата>*) – возвращают год, месяц, день месяца указанной даты.

Для формирования строковых выражений используется операция сцепления (конкатенации) обозначаемая знаком «+» и следующие функции:

– *CHARINDEX (<подстрока>, <строка> [, <начальная позиция>])* – возвращает позицию подстроки в строке. Поиск начинается с начальной позиции или с первого знака, если начальная позиция не указана. Если подстрока не найдена, функция возвращает ноль;

– *SUBSTRING (<строка>, <начало>, <длина>)* – возвращает подстроку указанной длины с указанного начального знака.

Есть функции, предназначенные для определения длины строки *LEN* преобразования прописных букв в строчные *LOWER* и наоборот *UPPER*, замены одной подстроки на другую *REPLACE* и др.

Следующие функции связаны с проверками значений:

– *ISNULL (<выражение>, <замена>)* – вместо выражение возвращается указанное значение замены, если значение выражения равно NULL;

– *COALESCE (<выражение> [, <выражение>]*) – возвращает первое не NULL-значение из списка;

– *CASE WHEN <условие> THEN <выражение>*

[WHEN <условие> THEN <выражение> ...]

[ELSE <else-выражение>] END – возвращает первое значение, для которого условие окажется истинным. Если такового не окажется, возвращается значение else-выражения.

Функция CONVERT(<тип данных>,<выражение>[,<тип даты>]) преобразует выражение к указанному типу.

4.5.3. Фильтрация строк

Условие из пункта WHERE должно содержать условия выбора строк – результата соединения таблиц (и условия соединения таблиц, если они не указаны в пункте FROM). Для формулировки условия могут использоваться срав-

нения и логические операции: AND – конъюнкция («логическое И»); OR – дизъюнкция («логическое ИЛИ»); NOT – отрицание.

Например, в запросе

```
SELECT S.Продавец, О.Дата, О.[Номер договора],
 Т.Товар, С.Количество, Т.[Единица измерения],
 Т.[Вес ЕдИзм(Кг)], С.Цена, С.Количество*С.Цена AS [Стоимость]
FROM Договоры О
 INNER JOIN Продавцы S ON О.[Код Продавца] = S.[Код продавца]
 INNER JOIN Продажи С
  ON О.[Номер договора] = С.[Номер договора]
 INNER JOIN Товары Т ON С.[Код товара] = Т.[Код товара]
WHERE (S.Город = 'Иркутск') AND (Т.Товар = 'Рис')
```

из базы данных извлекается информация о поставках риса из Иркутска.

Для формулировки условий применяются следующие специальные формы сравнений.

Условие *<выражение> [NOT] BETWEEN <начало> AND <конец>* истинно, если значение выражения лежит (не лежит в случае указания NOT) в указанном диапазоне. Например, для выделения договоров с номерами от 101 до 200 можно было бы воспользоваться условием *[Номер договора] BETWEEN 101 AND 200.*

Условие *< выражение > [NOT] LIKE <шаблон>* проверяет для значения выражения соответствие шаблону (несоответствие в случае указания NOT). В шаблоне можно использовать знаки: “_” (подчеркивание), означающий подстановку одного любого символа, “%”, заменяющий произвольную последовательность символов. Например, условие *Товар LIKE “Конфет%”* выбирает все товары, названия которых начинаются с «Конфет».

Следующие формы сравнения используют ***подзапросы*** *– запросы, вложенные в основной запрос для вычисления некоторых значений*. Подзапросы определяются командой SELECT.

Условие *<выражение> <сравнение> ALL (<подзапрос>)* истинно, если сравнение выполняется для всех данных, извлеченных подзапросом. Например, запрос

```
SELECT Продавцы.Продавец, Товары.Товар, Продажи.Количество,
 Товары.[Единица измерения], Продажи.Цена
FROM  Договоры
 INNER JOIN Продавцы
 ON Договоры.[Код Продавца] = Продавцы.[Код продавца]
 INNER JOIN  Продажи
 ON Договоры.[Номер договора] = Продажи.[Номер договора]
 INNER JOIN Товары
 ON Продажи.[Код товара] = Товары.[Код товара]
WHERE Продажи.Цена< ALL
(
SELECT С.Цена
FROM Договоры О
 INNER JOIN Продавцы S ON О.[Код Продавца] = S.[Код продавца]
```

```
 INNER JOIN Продажи C
ON O.[Номер договора] = C.[Номер договора]
WHERE (S.Продавец='Гермес') AND
 C.[Код товара] =  Продажи.[Код товара]
)
```

выбирает поставки, в которых цена товаров ниже, чем любые цены на такие же товары поставщика «Гермес».

Условие *<поле> <сравнение> ANY (<подзапрос>)* истинно, если сравнение выполняется хотя бы для одного значения, извлеченного подзапросом. Для извлечения поставок, в которых цена товара не превышает цен на такие же товары поставщика с 'Гермес'., можно воспользоваться предыдущим запросом, заменив в нем *ALL* на *ANY*.

Результат сравнения *EXIST (<подзапрос>)* – истина, если результат подзапроса является непустым множеством. Команда

```
SELECT *
FROM Товары
WHERE NOT EXISTS
(
SELECT *
FROM Договоры O
 INNER JOIN Продавцы S ON O.[Код Продавца] = S.[Код продавца]
 INNER JOIN Продажи C
ON O.[Номер договора] = C.[Номер договора]
WHERE (S.Продавец='Гермес') AND
 C.[Код товара] =  Товары.[Код товара]
)
```

выделят все товары, которые не поставлял «Гермес».

Сравнение *<поле> [NOT] IN (<набор значений>)* истинно, если значение поля входит (не входит, если указано NOT) в набор значений. Набор значений может быть указан перечислением, например *[Код продавца] IN (5, 7),* или подзапросом. Например, условие

```
[Код товара] IN
(
SELECT DISTINCT C.[Код товара]
FROM Договоры O
 INNER JOIN Продавцы S ON O.[Код Продавца] = S.[Код продавца]
 INNER JOIN Продажи C
ON O.[Номер договора] = C.[Номер договора]
WHERE (S.Продавец='Гермес')
)
```

означает принадлежность кода множеству кодов товаров, проданных «Гермесом».

4.5.4. Группировка

Фраза *GROUP BY* позволяет сгруппировать записи после объединения таблиц и фильтрации. Список выражений группировки (после фразы *GROUP BY*) состоит из выражений, разделенных запятыми. В группу попадают записи с одинаковыми значениями выражений группировки. В выходной таблице каждая группа представлена одной записью. По каждой группе можно вычислить агрегированные показатели. Запрос

```
SELECT Товары.Товар, SUM(Продажи.Количество) AS Количество,
 Товары.[Единица измерения],
 AVG(Продажи.Цена) AS [Средняя цена],
 SUM(Продажи.Количество*Продажи.Цена) AS Стоимость
FROM  Продажи INNER JOIN  Товары
ON Продажи.[Код товара] = Товары.[Код товара]
GROUP BY Товары.Товар, Товары.[Единица измерения]
```

выбирает таблицу, в которой для каждого товара вычисляется количество, средняя цена и стоимость по всем поставкам данного товара.

Условие выбора группы, указанное в пункте HAVING, определяет условие включения итогов вычисления по группе в результат запроса. Изменим запрос из предыдущего примера так, чтобы выбрать товары, для которых количество поставок было больше 5, добавив фразу *HAVING COUNT(*)>5*.

Условие выбора группы предназначено прежде всего для проверки агрегированных значений. Для уменьшения затрат на вычисление запроса рекомендуется обычные условия (не связанные с агрегированными значениями) указывать в пункте *WHERE*. В список колонок запроса с группировкой можно помещать только те выражения, которые представлены в списке полей группировки и результаты агрегирования.

Если фраза *GROUP BY* отсутствует, а список выражений, определяющих колонки таблицы-результата запроса, включает только константы и функции агрегирования, то вычисления выполняются по всем записям и результат будет представлен одной записью.

4.5.5. Сортировка записей результата

Записи выходной таблицы можно упорядочить, если в команду включить пункт ORDER BY. Элементом списка является выражение, за которым может следовать слово ASC для упорядочения по возрастанию или DESC для упорядочения по убыванию. Порядок записи выражений сортировки определяет последовательность сортировки: сначала выполняется сортировка по первому выражению, затем записи с одинаковым значением первого выражения сортируются по второму выражению и т.д.

4.5.6. Применение построителя запросов

Программа Management Studio предоставляет графические средства построения запросов. Запускается построитель запросов командой Design Query in Editor… в контекстном меню или меню Query. В окне построителя (Рис. 15) на

верхней панели с помощью контекстного меню можно добавлять таблицы, включать группировку, изменять тип запроса и выполнять другие команды. Основное назначение верхней панели – связывание операторами соединения, которые отображаются линией. Если определены ограничения ссылочной целостности, то они при добавлении таблицы сразу определяют соответствующее соединение. Контекстное меню соединения позволяет определять внешние соединения. Построение запроса можно контролировать, проверяя соответствующую команду на нижней панели построителя.

Средняя панель предназначена для определения колонок (Column), выбирая поля или вводя выражения. Кроме этого, можно задавать условия выбора в колонке Filter, отключать формирование колонки (Output), если выражение или поле используется только для фильтрации или группировки, задавать тип (Sort Type) и порядок (Sort Order) сортировки.

Включение с помощью контекстного меню группировки приводит к появлению колонки (Group By), в которой нужно для каждого формируемого поля выбрать группировку по нему (Group By), использование в фильтрации (Where) или применение функции агрегирования (Sum, или Count, или Avg, или другую).

Готовый запрос будет вставлен в окно Management Studio, из которого был вызван построитель запросов.

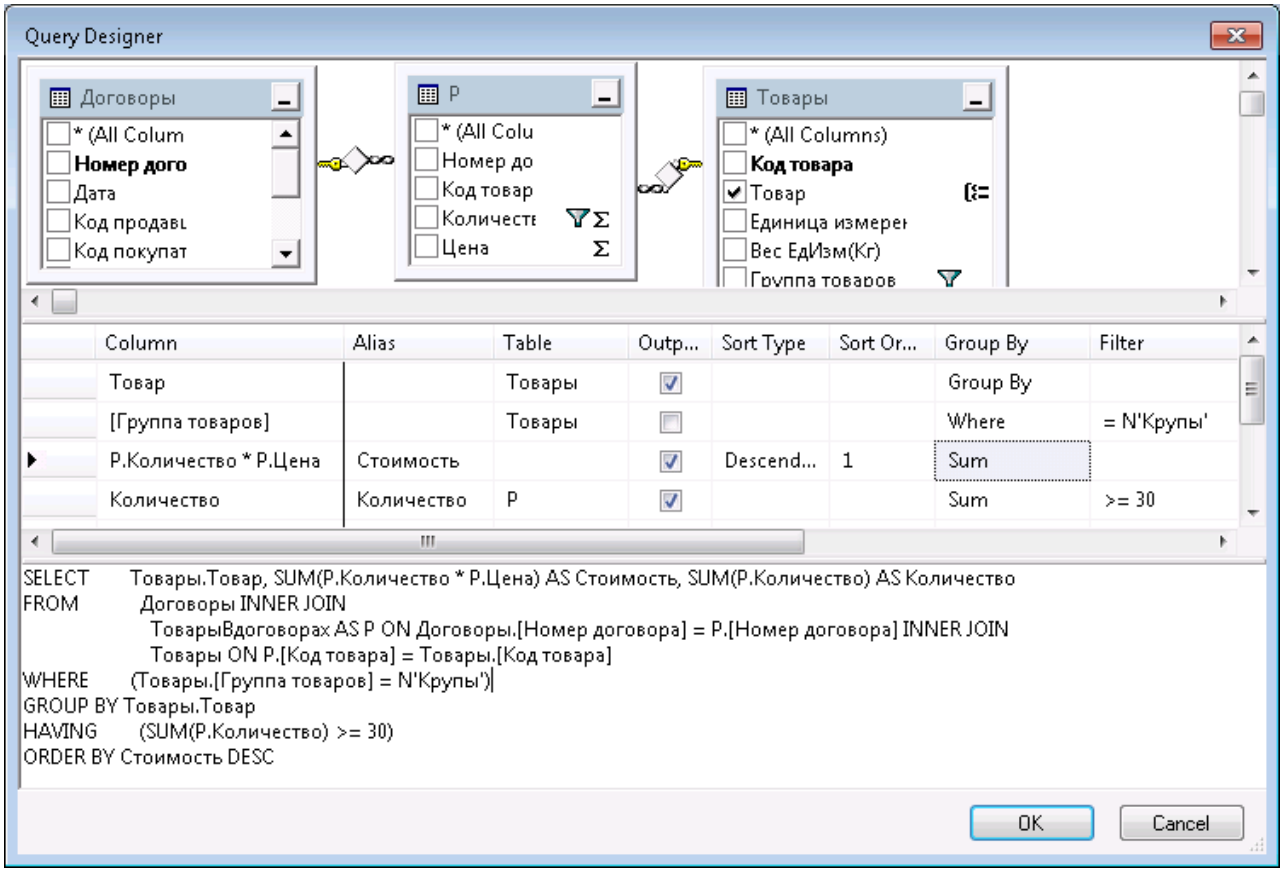


Рис. 15. Окно построителя запросов

4.5.7. Объединение записей нескольких запросов

Если структуры таблиц, возвращаемые несколькими запросами, одинаковы, то можно применять операторы, аналогичные операциям над множе-

ствами, а именно объединение (UNION), пересечение (INTERSECT) и разность (EXCEPT).

Для объединения записей таблиц предусмотрен оператор UNION

< запрос> UNION [ALL] <запрос>[UNION [ALL] <запрос>...] [ORDER BY <выражения сортировки>]

объединяющий записи всех запросов в одну таблицу. Запросы должны удовлетворять ограничениям:

– количество и порядок полей в запросах должны быть идентичными;

– типы данных полей должны быть сопоставимы.

Дублирующие записи вычеркиваются из объединения, если только не указано ключевое слово ALL. Если предусмотрена сортировка, то упорядочивается результат объединения.

В запросе

```
SELECT DISTINCT Товары.Товар
FROM Договоры INNER JOIN Продажи ON Договоры.[Номер договора] =
Продажи.[Номер договора]
 INNER JOIN Товары ON Продажи.[Код товара] = Товары.[Код товара]
WHERE (YEAR(Договоры.Дата) = 1997) AND (MONTH(Договоры.Дата) in
(6,7,8))
UNION
SELECT DISTINCT Товары.Товар
FROM Договоры INNER JOIN Продажи ON Договоры.[Номер договора] =
Продажи.[Номер договора]
 INNER JOIN Товары ON Продажи.[Код товара] = Товары.[Код товара]
WHERE Договоры.Дата between convert(datetime,'1.12.1997',104) and con-
vert(datetime,'28.02.1998',104)
```

первый SELECT выбирает товары, которые продавались летом 1997 г., второй – выбирает товары, которые продавались зимой 1997–1998 гг. Объединение формирует список товаров, которые продавались зимой или летом.

Пересечение

```
SELECT ...
INTERSECT
SELECT ...
```

выберет товары, которые продавались и зимой, и летом.

Разность

```
SELECT ...
EXCEPT
SELECT ...
```

выберет товары, которые продавались летом, но не продавались зимой.

4.5.8. Сохранение результата запроса

Наличие фразы *[INTO <таблица-результат>]* определяет сохранение таблицы-результата под указанным именем.

Обычно используют сохранение во временные таблицы локальные или глобальные. Имя локальной временной таблицы должно начинаться с одного

диеза («#»), имя глобальной – с двух диезов. Временные таблицы существуют на протяжении сеанса соединения сервера с пользователем (или в течение работы процедуры, функции или триггера, в которых они определены). Локальные таблицы доступны только в рамках сеанса (процедуры, функции или триггера), глобальные таблицы доступны всем пользователям. Временные таблицы и локальные, и глобальные автоматически уничтожаются по окончании сеанса связи (процедуры, функции, триггера).

Гораздо реже результат запроса сохраняется как обычная (не временная) таблица базы данных. Во-первых, для этого необходим специальный режим использования базы данных, во-вторых, пользователь, выполняющий такой запрос, должен иметь права на создание таблиц в базе данных, в-третьих, сохраненные данные могут устаревать при изменениях в БД.

4.5.9. Определение представлений пользователей

Представление пользователя *View* – поименованный запрос, который компилируется, оптимизируется и хранится в БД. Представление может использоваться вместо таблицы в командах выбора и изменения данных. Представления обычно применяются для хранения алгоритмов выбора данных. Кроме этого, представления являются средством достижения гибкости – при изменении схемы данных достаточно разработать представления для отображения «новых» таблиц в «старые», и избежать, таким образом, переделки «старых» запросов.

Представление пользователя View создается командой *CREATE VIEW <имя представления> [(<колонки через запятую>)] AS <запрос> [WITH CHECK OPTIONS]*.

Представление пользователя позволяет выполнять изменения (DELETE, UPDATE, INSERT), если выполнены следующие условия запроса-определения View:

- нет выбора уникальных значений (отсутствует ключ DISTINCT);
- в качестве колонок таблицы-результата используются только имена;
- в пункте FROM используется только одна таблица;
- при фильтрации в пункте WHERE не использует подзапросы;
- в запросе нет группировки.

Вариант WITH CHECK OPTIONS запрещает ввод данных, не удовлетворяющих условию WHERE.

4.5.10. Использование подзапросов

Запрос можно вкладывать в другой запрос. Такое вложение и будем называть подзапросом. Как уже упоминалось, подзапрос можно использовать в запросе в пункте FROM наравне с таблицами. Кроме этого, предусмотрены специальные условия с подзапросами. Наконец, если подзапрос возвращает одно значение, то его можно использовать как переменную или константу.

Приведенный ниже запрос строит список продавцов с количеством договоров выше среднего:

```
SELECT Продавцы.Продавец,COUNT(Договоры.[Номер договора]) AS [Кол-во договоров]
FROM   Договоры INNER JOIN Продавцы
          ON Договоры.[Код продавца] = Продавцы.[Код продавца]
GROUP BY Продавцы.Продавец
HAVING     COUNT(Договоры.[Номер договора]) >
 (
      SELECT AVG([Кол-во договоров])
            AS [Среднее кол-во договоров у продавца]
      FROM
      (
      SELECT     [Код продавца], COUNT(*) AS [Кол-во договоров]
      FROM       Договоры
      GROUP BY [Код продавца]
      ) КоличествоДоговоровПродавцов
 )
```

В запросе используется два подзапроса. Подзапрос *КоличествоДоговоровПродавцов* для каждого продавца находит количество договоров, которое следующий подзапрос усредняет и возвращает единственное значение.

4.6. Программные объекты в базе данных

Программные объекты – процедуры, функции, триггеры – являются важной частью БД. Вместе с представлениями пользователей (View) они позволяют описывать наиболее важные алгоритмы обработки данных. Хранение программ в БД имеет следующие преимущества по сравнению с реализацией обработки данных на стороне клиента:

– однократная реализация позволяет сократить расходы на разработку и модификацию алгоритмов;

– изменение программных объектов в БД не приводит к модификации программного обеспечения, работающего с базой;

– централизованное управление полномочиями в БД позволяет точно определить круг пользователей, которым разрешено использовать обработку и выбор данных, что повышает безопасность БД.

Для определения программных объектов вводятся специальные команды для создания переменных и описания обработки. К сожалению, каждый сервер предлагает свой вариант описания программ в БД. Поэтому перенести программные объекты с одной платформы на другую (например, из среды MS SQL server в среду Oracle) невозможно без переделки программ.

Для MS SQL server разработан специальный язык Transact-SQL, который является расширением языка SQL для разработки процедур, функций и триггеров. Команды этого языка позволяют определять сложные алгоритмы модификации и выборки данных.

Кроме описанных выше команд модификации и выбора данных, Transact-SQL включает следующие конструкции:

– Объявление локальных переменных (переменные в MS SQL должны начинаться с символа «@»)

DECLARE <переменная> <тип данных>

– Оператор присваивания

SELECT | SET <переменная>=<выражение>,…

– Операторные скобки (блок команд)

BEGIN
  <Команды>
END

– Проверка условия

IF <условие>
    {<команда> | <блок>}
[ELSE
    {<команда> | <блок>}]

– Цикл

WHILE <условие>
    {<команда> | <блок>}

Внутри блока команд цикла можно употреблять команду *BREAK* для выхода из цикла и команду *CONTINUE* для повторения цикла с начала (с проверки условия выполнения цикла).

– Безусловный переход

GOTO <метка>

выполняет переход на помеченный оператор

<метка>: <оператор>

– Комментарий

/ <Комментарий>*/*

-- <Комментарий>

– средства описания и вызова хранимых процедур, триггеров, функций

– команда выполнения команд SQL, заданных строковой переменной

Exec (<выражение строкового типа>)

Для получения результатов выполнения команд можно использовать системные функции, начинающиеся со знаков «@@», например:

– @@ERROR – код завершения последней команды (0 – нормальное завершение команды);

– @@ROWCOUNT – количество записей, обработанных последней командой.

4.7. Хранимые процедуры

Хранимые процедуры – это скомпилированные подпрограммы на языке Transact-SQL. Как обычно, процедура выполняет некоторую часто используемую обработку данных. Это может быть сложная корректировка данных или сложный выбор данных с применением нескольких временных таблиц. В последнем случае результатом вызова процедуры является сформированная фи-

нальной командой SELECT таблица. Этим хранимые процедуры SQL отличаются от процедур в других языках.

Процедура создается командой

CREATE PROCEDURE <имя процедуры> [(<@параметр> <тип>
[=<значение по умолчанию>] [OUTPUT], ...)] AS <команда> / <блок>.

Процедура вызывается командой

[[EXEC[UTE]] <имя процедуры>
[[<@параметр> =] {<значение> | <@переменная> [OUTPUT]].

Типичное применение процедур – это выполнение сложных корректировок: по полученным параметрам нужно выполнить сложные проверки и изменить данные нескольких таблиц в БД. Ниже приведен простой пример проверки в процедуре ссылочной целостности. Процедура NewOrder получает через параметры значения полей таблицы «Договоры», проверяет корректность ссылок «Код продавца» и «Код покупателя», добавляет запись с указанными значениями полей, определяет значение счетчика «Номер договора» в добавленной записи с помощью системной переменной @@IDENTITY и возвращает это значение через выходной параметр @N. Функция RAISERROR используется для формирования сообщения на русском языке в случае ошибки.

```
CREATE PROCEDURE NewOrder
      (@Date [datetime],
      @CodS [smallint],
      @CodP [smallint],
      @PayBefore [real],
      @N [smallint] OUTPUT)
AS
IF NOT EXISTS(SELECT * FROM [dbo].[Продавцы] WHERE [Код продавца]=@CodS)
  RAISERROR('В базе данных нет продавца с кодом указанным',16,-1)
ELSE
IF NOT EXISTS(SELECT * FROM [dbo].[Покупатели] WHERE [Код покупателя]=@CodP)
  RAISERROR('В базе данных нет покупателя с указанным кодом ',16,-1)
ELSE
BEGIN
INSERT INTO [dbo].[Договоры]
       ([Дата],[Код продавца],[Код покупателя],[Предоплата])
VALUES (@Date, @CodS, @CodP, @PayBefore)
SET @N = @@IDENTITY
END
```

Созданная процедура сохраняется в БД. И может быть вызвана следующими командами:

```
DECLARE @NewN [smallint]=0,
@d DateTime = convert(DateTime,'2.1.2000',104)
EXECUTE NewOrder @d, 1, 7, 0.1, @NewN OUTPUT
SELECT @NewN
```

Другое применение хранимых процедур заключается в выполнении сложной обработки, которая заканчивается формированием таблицы. Параметры процедуры при этом могут использоваться для указания условий выбора данных или уровня детализации агрегирования. Приведенная ниже процедура формирует сводный отчет по покупателям – товарам за указанный интервал дат с @d1 по @d2. Параметр @WithPollUp включает или отключает появление в результирующей таблице промежуточных итоговых записей для каждого уровня группировки (группировка с параметром WITH RollUP или без него).

```
CREATE PROCEDURE BuyerGoodsReport
(@d1 datetime, @d2 datetime, @WithPollUp bit=1) AS
IF @WithPollUp=1
        SELECT     b.Покупатель, t.[Группа товаров], t.Товар,
         MIN(t.[Единица измерения]) AS [Единица измерения],
         MIN(t.[Вес ЕдИзм(Кг)]) AS [Вес ЕдИзм(Кг)],
         SUM(td.Количество) AS СуммКоличество,
         SUM(td.Количество * td.Цена) AS СуммСтоимость,
         SUM(t.[Вес ЕдИзм(Кг)] * td.Количество) AS СуммВес
        FROM  Договоры d INNER JOIN Покупатели b
         ON d.[Код покупателя] = b.[Код покупателя]
         INNER JOIN ТоварыВдоговорах td
         ON d.[Номер договора] = td.[Номер договора]
         INNER JOIN Товары t ON td.[Код товара] = t.[Код товара]
        WHERE     (d.Дата BETWEEN @d1 AND @d2)
        GROUP BY b.Покупатель, t.[Группа товаров], t.Товар
        WITH RollUP
ELSE
        SELECT     b.Покупатель, t.[Группа товаров], t.Товар,
         MIN(t.[Единица измерения]) AS [Единица измерения],
         MIN(t.[Вес ЕдИзм(Кг)]) AS [Вес ЕдИзм(Кг)],
         SUM(td.Количество) AS СуммКоличество,
         SUM(td.Количество * td.Цена) AS СуммСтоимость,
         SUM(t.[Вес ЕдИзм(Кг)] * td.Количество) AS СуммВес
        FROM  Договоры d INNER JOIN Покупатели b
         ON d.[Код покупателя] = b.[Код покупателя]
         INNER JOIN ТоварыВдоговорах td
         ON d.[Номер договора] = td.[Номер договора]
         INNER JOIN Товары t ON td.[Код товара] = t.[Код товара]
        WHERE     (d.Дата BETWEEN @d1 AND @d2)
        GROUP BY b.Покупатель, t.[Группа товаров], t.Товар
```

Следующие команды вызывают данную процедуру для получения таблицы со сводными данными:

```
DECLARE          @d1 DateTime = convert(DateTime,'1.1.1997',104),
                 @d2 DateTime = convert(DateTime,'31.3.1997',104)
EXECUTE BuyerGoodsReport @d1, @d2, 1
```

Команда *INSERT <таблица> EXEC <процедура>* добавляет в таблицу записи, формируемые процедурой. В следующем примере создается временная таблица #Temp, в которую добавляются записи отчета за январь 1996 и 1997 гг.

```
DECLARE
        @d1 DateTime = convert(DateTime,'1.1.1996',104),
        @d2 DateTime = convert(DateTime,'31.1.1996',104),
        @d3 DateTime = convert(DateTime,'1.1.1997',104),
        @d4 DateTime = convert(DateTime,'31.1.1997',104)
CREATE TABLE #Temp(
    [Покупатель] [nvarchar](50) NULL,
    [Группа товаров] [nvarchar](30) NULL,
    [Товар] [nvarchar](50) NULL,
    [Единица измерения] [nvarchar](10) NULL,
    [Вес ЕдИзм(Кг)] [real] NULL,
    [СуммКоличество] [int] NULL,
    [СуммСтоимость] [float] NULL,
    [СуммВес] [float] NULL
)
INSERT #Temp
EXECUTE BuyerGoodsReport @d1, @d2, 0
INSERT #Temp
EXECUTE BuyerGoodsReport @d3, @d4, 0
SELECT * FROM #Temp
```

4.8. Триггеры

Триггер – процедура без параметров, определяемая для операции Update, Insert, Delete или их комбинации над определенной таблицей. Команда определения триггера имеет следующий формат:

```
CREATE TRIGGER <имя триггера>
    ON { <имя таблицы>| <имя представления>}
    { FOR | AFTER | INSTEAD OF }
     {[DELETE] [,] [INSERT] [,] [UPDATE]}
        AS
        <команды Transact – SQL >
```

Триггер запускается при выполнении перечисленных в его определении изменений до их фиксации в базе данных (триггер For или After) или вместо фиксации (триггер Instead of). Для программирования изменений в зависимости от произведенных действий в теле триггера можно использовать две таблицы:

– *Deleted* – содержит удаляемые или изменяемые записи (данные до операции);

– *Inserted* – содержит измененные или вставленные записи (данные после операции).

До выполнения операции проверяются ограничения целостности, которые могут и не допустить выполнения триггера. Следующий триггер For

```
CREATE TRIGGER КаскадноеУдалениеДоговора ON [Договоры]
FOR DELETE
AS
  DELETE FROM Продажи WHERE [Номер договора] IN
      (SELECT [Номер договора] FROM Deleted)
```

выполняющий каскадное удаление из таблицы «Продажи» записей, связанных с удаляемыми договорами, не будет выполнен, если определен внешний ключ для таблицы «Продажи» на таблицу «Договоры» и есть записи, ссылающиеся на удаляемый договор.

При наличии упомянутого внешнего ключа можно воспользоваться триггером Instead of:

```
CREATE TRIGGER КаскадноеУдалениеДоговора ON [Договоры]
INSTEAD OF DELETE
AS
  DELETE FROM Продажи WHERE [Номер договора] IN
      (SELECT DISTINCT [Номер договора] FROM Deleted)
  DELETE FROM Договоры WHERE [Номер договора] IN
      (SELECT DISTINCT [Номер договора] FROM Deleted)
```

В этом случае приходится программировать удаление записей из таблицы «Договоры» (вторая команда DELETE), так как команда пользователя по удалению договоров не выполняется, а только моделируется: записи, определенные пользователем для удаления, помещаются в таблицу Deleted.

Для проверки изменения некоторого поля применяется условие *IF UPDATE (<имя поля>)* которое возвращает значение «истина», если указанное поле было изменено командой UPDATE.

Триггеры являются эффективным средством для поддержки бизнес-правил. Например, при проведении бухгалтерских операций должно выполняться равенство сумм по дебету и кредиту. Такие правила не могут быть реализованы на основе механизма ссылочной целостности. Для регистрации и проводки хозяйственной операции в таблице «Операции» можно разработать триггер, который будет выполнять проводки, модифицируя соответствующие счета.

Триггеры допускают вложенный вызов. Например, при удалении поставщика запускается триггер, который удаляет договоры данного поставщика. Удаление договоров вызывает, в свою очередь, триггер для удаления продаж, включенных в удаляемые договоры.

Рекурсивный вызов триггера означает, что триггер определяет модификации, которые снова приводят к его вызову. Рекурсивный вызов возможен при включении соответствующего параметра базы данных.

4.9. Отображение программных объектов в Management Studio

Программа Management Studio демонстрирует программные объекты в обозревателе объектов (Рис. 16). Триггеры являются частью определения таблиц и отображаются в соответствующем разделе объектного описания таблицы. Представления пользователя – поименованные запросы – собраны в разделе

Views, хранимые процедуры и функции – в соответствующих подразделах Programmability.

Для каждого программного объекта с помощью контекстного меню можно получить скрипт его создания или перейти в режим его корректировки.

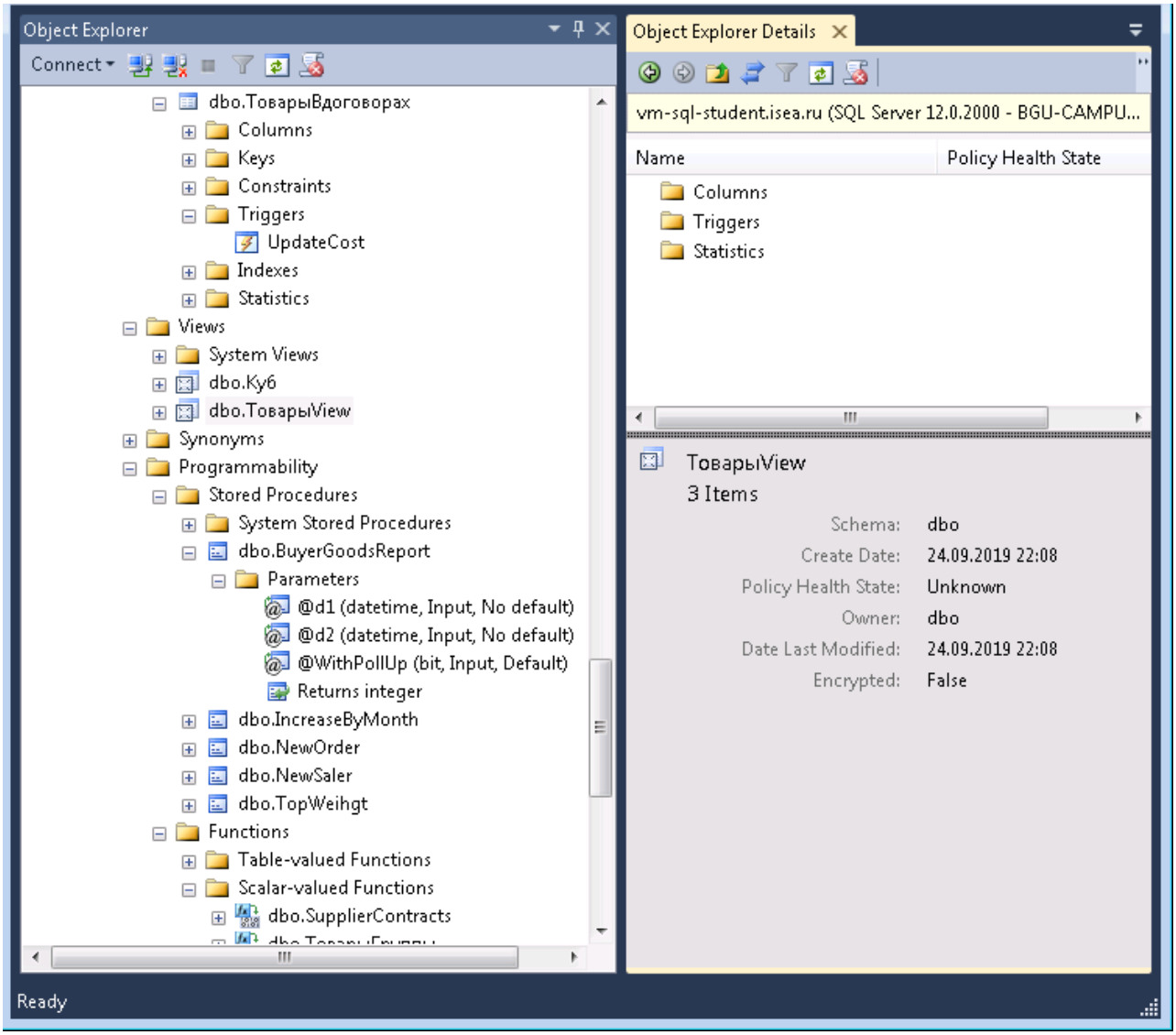


Рис. 16. Обозреватель объектов базы данных

4.10. Курсоры

Курсоры предоставляют возможность обрабатывать таблицы по записям. Существует определенная схема использования курсоров:

1. Сначала курсор надо объявить:

DECLARE <имя курсора> CURSOR FOR <команда выбора select>.

2. Для использования курсора он открывается *OPEN <имя курсора>*. При этом выполняется команда SELECT курсора и сохраняется ее результат – временная таблица, которую можно просматривать.

3. Просматривается курсор по записям обычно в цикле. Команда просмотра выполняет позиционирование записи и извлечение полей в переменные

FETCH {NEXT | PRIOR | FIRST | LAST | ABSOLUTE <номер записи> | RELATIVE <смещение>} FROM <имя курсора> INTO <@переменная_1>,..., <@переменная_n>, где *n* – число полей. Команда извлекает поля указанной записи и присваивает указанным переменным. В цикле просмотра записей обычно используется глобальная переменная @@FETCH_STATUS, которая содержит код завершения команды FETCH:

- 0 – успешно;
- –1 – конец таблицы;
- –2 – прочитана удаленная запись.

4. После окончания обработки записей курсора необходимо закрыть временную таблицу командой *CLOSE <имя курсора>*. При этом снимаются все блокировки, освобождается таблица курсора и ее снова можно открыть командой *OPEN*.

5. Удаление ссылки на курсор: *DEALLOCATE <имя курсора>*. При удалении ссылки на курсор освобождается имя переменной.

Пример:

```
/*******************
  Процедура вычисляет изменение объема продаж
  в процентах по отношению к предыдущему месяцу
*******************/
CREATE PROCEDURE IncreaseByMonth AS
BEGIN
-- Временная таблица для хранения вычислений
CREATE TABLE #I
(       Year int,
        Month int,
        Cost float,
        Increment float)
-- Курсор для вычисления объемов продаж по месяцам
DECLARE Costs CURSOR FOR
SELECT Year(Договоры.Дата) AS [Year],
   Month(Договоры.Дата) AS [Month],
   SUM(Количество*Цена) AS Cost
FROM Продажи INNER JOIN Договоры
   ON Продажи.[Номер договора] = Договоры.[Номер договора]
GROUP BY Year(Договоры.Дата), Month(Договоры.Дата)
ORDER BY Year(Договоры.Дата), Month(Договоры.Дата)
-- Открывается курсор – вычисляется запрос
OPEN Costs
-- Объявление переменных
DECLARE @Y int, @M int, @C0 float, @C float
-- извлечение первой записи курсора
FETCH NEXT FROM Costs INTO @Y, @M, @C0
-- формирование первой записи результата
INSERT #I VALUES( @Y, @M, @C0, NULL)
```

```
-- Цикл по записям курсора
FETCH NEXT FROM Costs INTO @Y, @M, @C
WHILE (@@FETCH_STATUS <> -1)
BEGIN
    -- формирование очередной записи результата
    INSERT #I
    VALUES( @Y, @M, @C,(@C-@C0)/@C0*100)
    -- извлечение очередной записи курсора
    FETCH NEXT FROM Costs INTO @Y, @M, @C
END
-- закрытие курсора
CLOSE Costs
-- Ликвидация курсора
DEALLOCATE Costs
-- Вывод результата
SELECT * FROM #I
END
```

4.11. Переменные табличного типа и функции

В MS SQL Server 2000 появились новые средства для увеличения гибкости и выразительности языка. В нем можно использовать локальные переменные, которые имеют табличный тип:

DECLARE <@переменная> TABLE (<определение полей и ограничений как в команде CREATE TABLE>)

Областью действия переменной, как обычно, являются процедура, функция или пакет команд.

Эти нововведения позволили определять функции, которые возвращают таблицы. Определение табличной функции с помощью нескольких команд выглядит следующим образом:

```
CREATE FUNCTION <имя функции>
  ( [<@параметр> [AS] <тип> [ = <значение по умолчанию>] [,...]] )
RETURNS <@переменная> TABLE <определение таблицы>
 [ AS ]
BEGIN
  <команды>
   RETURN
END
```

Следующая функция заменяет систему представлений – одно представление для каждого города поставщика:

```
CREATE FUNCTION ПродавцыИзГорода (@Town nVarChar(50))
RETURNS @s TABLE (
    [Код продавца] [int] NOT NULL ,
    [Форма собственности] [nvarchar] (10) NULL ,
    [Продавец] [nvarchar] (30) NULL ,
    [Город] [nvarchar] (30) AS NULL ,
```

```
    [Банк] [nvarchar] (30) NULL) AS
BEGIN
    INSERT @s
    SELECT * FROM Продавцы Where Город=@Town
    RETURN
END
```

Вызов функции, вычисляющей таблицу, возможен вместо употребления имени таблицы, например *select * from dbo.ПродавцыИзГорода ('Иркутск')*.

4.12. Ограничения языка SQL

Простота реляционной модели является основой ее эффективности. Она же определяет ее основные ограничения.

Прежде всего страдает выразительность языка – в нем нет агрегатов (обобщений). Нельзя объединить набор атрибутов в единое понятие (например, адрес) и манипулировать им как единым целым.

Проблематично в SQL напрямую реализовывать современные тенденции, например объектное описание данных, отчасти по причине отсутствия обобщений, отчасти по причине отсутствия средств наследования и инкапсуляции.

Ряд проблем создает ограниченность самого языка. Можно найти максимальную стоимость продаж, но найти продавца с максимальной стоимостью продаж алгоритмически сложнее. Другой проблемой SQL является невыразимость транзитивного замыкания, т.е. невозможно построить запрос, когда нужно найти для определенного элемента все связанные с ним элементы. Например, всех вышестоящих начальников, а не только непосредственного, всех потомков, а не только детей, и т.д. В таких запросах количество соединений является неопределенным, что могло бы быть реализовано отсутствующим в SQL рекурсивном соединении.

Вопросы

1. Почему SQL является декларативным языком?
2. Перечислите типы данных языка SQL и укажите область применения каждого.
3. Что включает определение таблицы в языке SQL?
4. Перечислите и определите свойства и ограничения колонок. Приведите примеры.
5. Почему пустые значения могут привести к ошибкам?
6. Перечислите и определите табличные ограничения. Приведите примеры.
7. Какие средства предлагает язык SQL для поддержания ссылочной целостности и реализации бизнес-правил?
8. Что такое индекс таблицы? Зачем и когда создаются индексы таблицы?
9. Перечислите и определите команды изменения содержания таблицы. Приведите примеры.
10. Перечислите основные фразы команды SELECT. Что, в какой форме, на основании чего выбирает эта команда?

11. Как определяются поля выбираемой таблицы в команде SELECT?

12. Перечислите типы выражений в команде SELECT, укажите правила их построения.

13. Для чего и как применяются статистические функции (функции агрегирования) в команде SELECT?

14. Перечислите правила и возможности соединения таблиц в команде SELECT.

15. Как выполняется фильтрация в команде SELECT?

16. Перечислите условия с подзапросами. Приведите примеры.

17. Как определяется группировка в команде SELECT? Что и как группируется? Что является результатом группировки?

18. В чем разница условий, задаваемых в пунктах *WHERE* и *HAVING*?

19. Перечислите возможности сортировки в команде SELECT.

20. Перечислите операции по объединению в одну таблицу результатов нескольких запросов.

21. Укажите назначение и использование представлений пользователя в языке SQL.

22. Для чего используется язык Transact-SQL? Какие команды он включает?

23. Укажите назначение и использование хранимых процедур, триггеров, курсоров и функций в языке SQL.

24. Как в теле триггера можно определить, какие изменения выполнены пользователем?

25. Когда запускается триггер?

5. Технология «клиент-сервер»

5.1. Технология и модели сетевой обработки данных

«Клиент-сервер» – это модель взаимодействия компьютеров в сети. Компьютер (процесс), управляющий тем или иным ресурсом, принято называть сервером этого ресурса, а компьютер (процесс), использующий ресурс, – клиентом.

В технологии «клиент-сервер» в качестве сервера подразумевается сервер баз данных. Взаимодействие клиента и сервера представлено на рис. 17. Клиентская программа преобразует действия клиента в команду SQL и передает команду посредством клиентской части SQL-сервера и сетевых библиотек в сеть. Сервер через сетевые библиотеки получает команду и передает ее ядру SQL-сервера, которое и выполняет SQL-команды и обратным порядком отправляет результат клиенту для отображения результата на компьютере пользователя.

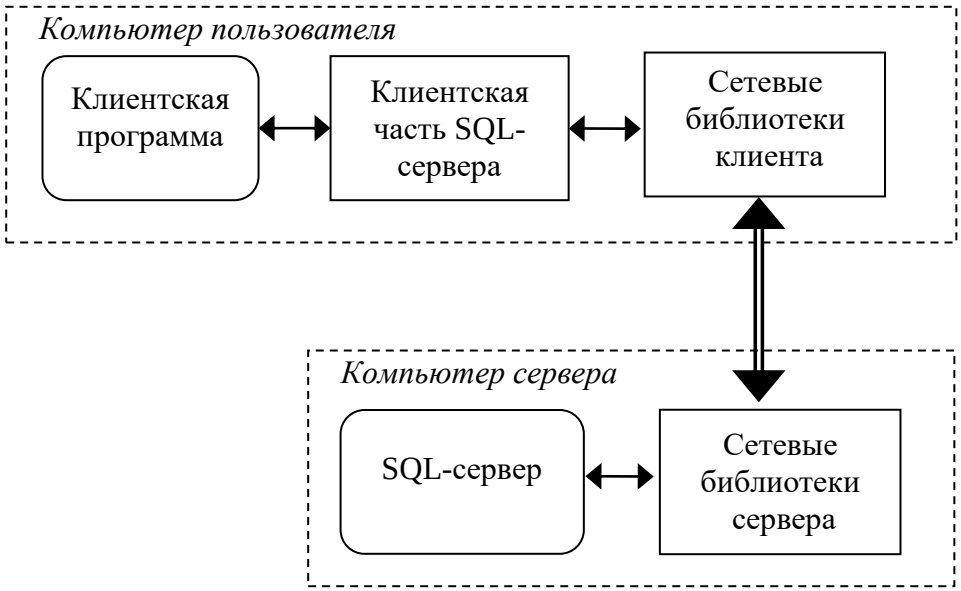


Рис. 17. Взаимодействие клиента и сервера баз данных
в технологии «клиент-сервер»

В технологии «клиент-сервер» часть функций прикладной программы реализована в программе-клиенте, другая – в программе-сервере. Функции доступа к данным можно разделить на три группы:

– презентационная логика – включает интерфейсные функции (задание команд и получение результатов их выполнения) – реализуется на клиенте;

– бизнес-логика – правила корректировки и обработки данных, вытекающие из логики учета и управления – может быть реализована как на клиенте, так и на сервере;

– логика доступа – функции доступа к данным – реализуется сервером.

Варианты архитектуры, основанные на делении работ между клиентом и серверами, представлены в табл. 1.

Таблица 1

Многоуровневые модели «клиент-сервер»

<table>
<tr><td rowspan="2">Уровни логики</td><td rowspan="2">«Файл-сервер»</td><td colspan="2">«Клиент-сервер»</td></tr>
<tr><td>Толстый клиент</td><td>Тонкий клиент</td></tr>
<tr><td>Презентационная логика</td><td rowspan="3">Клиент</td><td rowspan="2">Клиент</td><td>Клиент</td></tr>
<tr><td>Бизнес-логика</td><td rowspan="2">Сервер приложений
Сервер БД</td></tr>
<tr><td>Логика доступа к ресурсам</td><td>Сервер БД</td></tr>
</table>

В модели «файл-сервер» в приложении совмещены все виды логики. Протокол обмена представляет собой набор низкоуровневых вызовов, обеспечивающих приложению доступ к файловой системе на файл-сервере.

В модели «клиент-сервер» БД презентационная логика и часть бизнес-логики совмещены и выполняются на компьютере-клиенте. Другая, общезначимая часть бизнес-логики оформлена как набор хранимых процедур и триггеров и функционирует на компьютере-сервере БД. Доступ к информационным ресурсам обеспечивается операторами языка SQL.

Реализация бизнес-логики на сервере имеет следующие достоинства (знак «+») и недостатки (знак «–»):

+ гарантия выполнения,

+ доступ к правилам и их использование всеми клиентами,

+ быстродействие за счет применения мощных серверов и уменьшения трафика сети,

+ мобильность,

+ удобство администрирования,

– правила не могут использовать несколько БД,

– сложность интерпретации сообщений сервера об ошибках.

Достоинства и недостатки реализации бизнес-логики в программе-клиенте в определенном смысле противоположны предыдущему варианту:

+ возможен высокий уровень оперативного контроля, использующий в том числе и диалог с пользователем,

– замедление работы клиента и увеличение трафика сети за счет передачи данных клиенту для обработки,

– повторная реализация бизнес-правил для разных приложений-клиентов,

– неудобно собирать все бизнес-правила в один свод, так как они разбросаны по всему тексту приложения или по текстам нескольких приложений,

– незащищенность БД от изменений без соблюдения правил некорректно разработанными клиентами.

Выбор, конечно, зависит от конкретной ситуации, однако чаще всего выбирается реализация бизнес-логики на сервере по причинам бóльших преимуществ такой архитектуры.

5.2. Обработка транзакций

Для описания единицы работы в БД вводится понятие транзакции. *Транзакция* – это логическая единица работы с базой данных. Для SQL-сервера транзакция представляет собой последовательность операторов языка SQL, которая рассматривается как некоторое неделимое действие над базой данных.

Транзакция характеризуется четырьмя классическими свойствами ACID (Atomicity, Consistency, Isolation, Durability): атомарности, согласованности, изолированности, долговечности (прочности):

1. Свойство ***атомарности*** выражается в том, что транзакция должна быть выполнена целиком или не выполнена вовсе.

2. Свойство ***согласованности*** гарантирует, что выполнение транзакции переводит данные из одного согласованного состояния в другое – транзакция не разрушает согласованности данных, хотя может ее нарушать в процессе выполнения.

3. Свойство ***изолированности*** означает, что результат выполнения транзакции не зависит от выполняющихся в то же время других транзакций. Для этого конкурирующие за доступ к данных транзакции обрабатываются по определенным правилам.

4. Свойство ***долговечности*** трактуется следующим образом: если транзакция завершена успешно, то те изменения в данных, которые были ею произведены, не могут быть потеряны в результате сбоев.

Команда: *BEGIN TRAN[ACTION]* начинает транзакцию – все последующие команды образуют блок команд транзакции. Заканчивается транзакция командой *COMMIT TRAN[SACTION]* или *COMMIT WORK*, которая предписывает серверу зафиксировать все изменения, выполненные командами транзакции. Второй вариант – завершение транзакции командой *ROLLBACK TRAN[SACTION]* или *ROLLBACK WORK* заставляет сервер отменить все изменения, сделанные транзакцией.

Сервер работает по умолчанию в режиме автоматического начала транзакций, в котором каждая команда рассматривается как транзакция, если нет явных команд определения транзакции. Этот режим устанавливается командой *SET IMPLICIT_TRANSACTION OFF*.

Другой режим *неявного начала транзакций* (устанавливается командой *SET IMPLICIT_TRANSACTION ON*) автоматически начинает новую транзакцию, если закончена предыдущая командой фиксации или отката.

Откат и фиксация транзакций становятся возможными благодаря журналу транзакций. При выполнении любого оператора SQL, который вносит изменения в базу данных, СУБД автоматически заносит сведения об этом в журнал транзакций. Только после записи в журнал транзакций СУБД действительно вносит изменения в базу данных. Если после очередного оператора SQL был выполнен оператор COMMIT, то в журнале транзакций делается отметка о завершении текущей транзакции. Если же после оператора SQL следовал оператор ROLLBACK, то СУБД просматривает журнал транзакций и отыскивает в нем записи, отражающие состояние измененных строк до внесения изменений.

Используя их, СУБД восстанавливает те строки в таблицах базы данных, которые были изменены текущей транзакцией, и таким образом отменяет все изменения, выполненные транзакцией.

Одной из основных задач многопользовательских СУБД является организация одновременного доступа к одним и тем же данным множества пользователей. Проблемы коллективного доступа разбивают на следующие группы: потеря изменений, чтение грязных данных (незафиксированных изменений), неповторяющееся чтение, появление «данных-фантомов».

Потеря изменений происходит в ситуации, когда две или несколько транзакций читают одни и те же данные из базы данных, вносят в них какие-либо изменения и затем пытаются одновременно записать результат по прежнему месту. Разумеется, в базе данных могут быть сохранены изменения, выполненные только одной транзакцией. Другие изменения будут потеряны.

Проблема чтения «грязных» данных (незафиксированных изменений) возникает в случае, когда в процессе выполнения одной транзакции в данные были внесены изменения и тут же прочитаны другой транзакцией, однако затем в первой программе транзакция была прервана оператором ROLLBACK. Получается, что вторая программа прочитала неверные, «грязные», незафиксированные данные.

Неповторяющееся чтение встречается, когда первая транзакция выполняет повторное чтение данных, которые были измены второй транзакцией, между первым и вторым чтением. Это повторное чтение первой транзакцией считывает данные, отличные от прочитанных в первый раз.

Появление «данных-фантомов» возникает, когда одна транзакция читает данные, а другая добавляет данные. Добавленные данные являются невидимыми – «фантомами» – для первой транзакции.

Стандарт ANSI предусматривает четыре уровня изолированности транзакций:

1. ***Read Uncommited***. Нулевой уровень изолированности – запрет потерянных изменений. Не допускается обновление данных, пока не закончится первая транзакция, обновляющая эти данные.

2. ***Read Commited***. Первый уровень изолированности – нет потерянных и незафиксированных изменений и «грязного чтения». Не допускается чтение данных, пока не закончится первая транзакция, обновляющая эти данные.

3. ***Repeatable Read***. Второй уровень изолированности – запрещено неповторяемое чтение незафиксированных данных и потерянные изменения. Не допускается обновление и чтение данных, пока не закончится транзакция, прочитавшая эти данные.

4. ***Serializable***. Третий уровень изолированности – нет фантомов и других конфликтов. Не допускается обновление и чтение таблиц, пока не закончится транзакция, обновляющая или читающая эти таблицы.

Уровень изолированности транзакций устанавливается командой:

SET TRANSACTION ISOLATION LEVEL
{ READ UNCOMMITED
READ COMMITED

REPEATABLE READ
SERILIZABLE }

Автоматический откат транзакции устанавливается командой *SET XACT_ABORT ON.*

При возникновении ошибки будет произведен откат всех команд транзакции. Если автоматический откат транзакций отключен, то будет отменена только последняя команда, которая вызвала ошибку.

Чем выше уровень изолированности, тем меньше возможностей для параллельного выполнения транзакций. Основным механизмом достижения изолированности является механизм блокировок. Транзакция запрашивает захват объекта в одном из режимов: совместный (S – Shared) или монопольный (X – eXclusive). Для реляционных БД объектами захвата могут быть:

– база данных;
– файл: несколько отношений с индексами;
– отношение;
– страница данных: несколько кортежей, индексная страница;
– запись.

Чем крупнее объект, тем меньше возможностей параллельного выполнения транзакций и меньше расходы на управление блокировками.

Одной из проблем механизма блокирования является «тупик» или «мертвая блокировка» – DeadLock. «Тупик» возникает, если две транзакции захватили разные объекты и каждая пытается захватить объект, заблокированный другой транзакцией. Сервер БД имеет встроенный механизм разрешения этой проблемы, если блокировки не заданы явно специальной командой.

5.3. Обработка распределенных данных

Главная проблема больших систем – организация обработки распределенных данных. Данные могут находиться на серверах различных моделей и производителей, функционирующих под управлением различных операционных систем, а доступ к данным может осуществляться разнородным программным обеспечением. Сами компьютеры могут быть территориально удалены друг от друга и находиться в различных географических точках. Решение этой проблемы предлагают две технологии: технология распределенных баз данных (транзакций) и технология тиражирования данных.

Технология распределенных баз данных

Под распределенной базой данных подразумевают базу данных, включающую фрагменты, которые располагаются на различных серверах компьютерной сети, и, возможно, управляются различными СУБД. Тем не менее распределенная база данных должна выглядеть с точки зрения пользователей и прикладных программ как обычная база данных.

В распределенных базах транзакция, выполнение которой заключается в обновлении данных на нескольких серверах сети, называется глобальной или распределенной транзакцией.

Для пользователя распределенной базы данных глобальная транзакция выглядит как обычная. Для реализации этой технологии в современных СУБД предусмотрен так называемый протокол двухфазной фиксации транзакций.

Фаза 1 начинается, когда транзакция фиксируется. Специальный монитор распределенных транзакций направляет уведомление «Подготовиться к фиксации» всем серверам, выполняющим распределенную транзакцию. Если все серверы приготовились к фиксации, монитор распределенных транзакций принимает решение о фиксации. Если хотя бы один из серверов не откликнулся на уведомление в силу каких-либо причин, будь то аппаратная или программная ошибка, то монитор распределенных транзакций откатывает транзакции на всех серверах, включая даже те, которые подготовились к фиксации и оповестили его об этом.

Фаза 2 – монитор распределенных транзакций направляет команду «Зафиксировать» всем серверам, затронутым транзакцией, и гарантирует, что транзакции на них будут зафиксированы. Если связь с сервером потеряна в интервал времени между моментом, когда монитор распределенных транзакций принимает решение о фиксации транзакции, и моментом, когда сервер подчиняется его команде, то монитор распределенных транзакций продолжает попытки завершить транзакцию, пока связь не будет восстановлена.

Технология тиражирования данных

Принципиальное отличие технологии тиражирования данных от технологии распределенных баз данных заключается в отказе от синхронной обработки транзакции всеми серверами. Ее суть состоит в том, что любая БД (как для СУБД, так и для работающих с ней пользователей) всегда является независимой от данных, расположенных на других серверах. Все транзакции в системе завершаются на одном сервере. Это приводит к хранению на каждом сервере своей копии общих данных и периодической синхронизации изменений, внесенных на разных серверах.

Тиражирование данных – это асинхронный перенос изменений исходной базы данных в базы данных, размещенные на различных серверах распределенной системы. Функции тиражирования данных выполняет специальный модуль СУБД – сервер тиражирования данных, называемый репликатором. Его задача – поддержка идентичности данных в принимающих базах данных данным в исходной базе.

В качестве базиса для тиражирования выступает транзакция. В то же время возможен перенос изменений группами транзакций, периодически или в некоторый момент времени.

Технология распределенных БД и технология тиражирования данных – в определенном смысле антиподы. Краеугольный камень первой – синхронное завершение транзакций одновременно на нескольких узлах распределенной системы, т.е. синхронная фиксация изменений в распределенной БД. «Ахиллесова пята» этой технологии – жесткие требования к производительности и надежности каналов связи и серверов. Если БД распределена по нескольким территориально удаленным узлам, объединенным медленными и ненадежными каналами

связи, а число одновременно работающих пользователей составляет десятки и выше, то вероятность того, что распределенная транзакция будет зафиксирована в обозримом временном интервале, становится чрезвычайно малой. В таких условиях обработка распределенных данных практически невозможна.

Реальной альтернативой технологии распределенных БД становится технология тиражирования данных, не требующая синхронной фиксации изменений (и в этом ее сильная сторона). В действительности далеко не во всех задачах требуется обеспечение идентичности БД на различных узлах в любое время. Достаточно поддерживать тождественность данных лишь в определенные критичные моменты времени. Следовательно, можно накапливать изменения в данных в виде транзакций в одном узле и периодически копировать эти изменения на другие узлы.

Просуммируем очевидные преимущества технологии тиражирования данных. Во-первых, данные всегда расположены там, где они обрабатываются, следовательно, скорость доступа к ним существенно увеличивается. Во-вторых, передача только операций, изменяющих данные (а не всех операций доступа к удаленным данным, как в технологии распределенных БД), и к тому же в асинхронном режиме, позволяет значительно уменьшить трафик. В-третьих, со стороны исходной БД для принимающих БД репликатор выступает как процесс, инициированный одним пользователем, в то время как в физически распределенной среде с каждым локальным сервером работают все пользователи распределенной системы, конкурирующие за ресурсы друг с другом. Наконец, в-четвертых, никакой продолжительный сбой связи не в состоянии нарушить передачу изменений. После восстановления связи передача возобновляется с той транзакции, на которой тиражирование было прервано.

Технология тиражирования данных не лишена некоторых недостатков, вытекающих из ее специфики. Например, невозможно полностью исключить конфликты между двумя версиями одной и той же записи. Он может возникнуть, когда вследствие все той же асинхронности два пользователя на разных узлах исправят одну и ту же запись в тот момент, пока изменения в данных из первой базы данных еще не были перенесены во вторую. Следовательно, при проектировании распределенной среды с использованием технологии тиражирования данных необходимо предусмотреть конфликтные ситуации и запрограммировать репликатор на какой-либо вариант их разрешения.

Вопросы

1. Перечислите и укажите назначение видов логики в сетевой системе регистрации и обработки данных. Где может быть реализован каждый вид логики в системах «файл-сервер» и «клиент-сервер»?

2. Перечислите достоинства и недостатки реализации бизнес-логики на стороне клиента и на стороне сервера.

3. Понятие и свойства транзакции.

4. Перечислите и опишите использование команд начала и завершения транзакций.

5. Как реализуется механизм отката транзакций?

6. Опишите основные проблемы коллективного доступа к данным.

7. Перечислите и опишите уровни изолированности пользователей. Какие проблемы коллективного доступа они решают?

8. Как и что блокируется транзакциями в БД для обеспечения каждого уровня изолированности?

9. Опишите основные принципы, достоинства и недостатки технологий распределенных БД и тиражирования.

6. MS SQL сервер. Общие сведения

6.1. Компоненты MS SQL сервер

MS SQL сервер работает в среде операционной системы MS NT server и реализован в виде нескольких сетевых служб, основными среди них являются:

– MSSQLServer – основное ядро сервера, реализует функции доступа к данным, только эта служба необходима для доступа к данным;

– SQLServerAgent обеспечивает автоматическое выполнение заданий, извещение персонала об ошибках;

– Analysis Services – аналитические службы реализуют технологию многомерного анализа и алгоритмы исследования зависимостей (Data Mining);

– Integration Services – службы, интегрирующие данные из разных источников;

– Reporting Services (SSRS) – служба создания отчетов.

Каждая служба сервера запускается под некоторой учетной записью. Рекомендуется такие учетные записи включать в группу Administrators для автоматического запуска и давать учетным записям права Log on as a service, Act as a part of the operating system.

Предусмотрены следующие системные БД:

– Master – хранение параметров конфигурации сервера, login-ов пользователей и других системных данных;

– Msdb – используется SQLServerAgent для хранения информации о заданиях, операторах и оповещениях;

– Model – шаблон БД, который копируется в каждую создаваемую БД;

– Tempdb – БД для хранения временных таблиц.

Любая БД содержит системные таблицы для хранения метаданных.

Каждая БД может хранить схему с предоставлением доступа к таблице через имя схемы имя_схемы.имя_таблицы.

Синоним переносит идею указания таблицы через имя схемы на любые другие объекты БД. Синоним заменяет любую квалифицированную ссылку.

Взаимодействие клиента и MS SQL сервер выполняется через специальные интерфейсы программирования (API) доступа к базам данных:

– ODBC (Open Database Connectivity) – набор функций для доступа к табличным источникам данных. ODBC был использован для построения более совершенных и удобных драйверов Remote Data Objects (RDO), DataAccess Objects (DAO);

– DB-Library – библиотека для доступа к серверу;

– OLE DB – новая открытая спецификация для доступа к данным;

– ADO (ActiveX Data Object) – эволюционное развитие RDO и DAO.

Все API реализованы в виде dll-файлов, которые взаимодействуют с MS SQL сервер через сетевые библиотеки клиента и сервера.

Службы сервера используют следующие каталоги:

– Backup – резервное копирование;

– Bin – программы;

– Data – базы данных;
– Jobs – временные файлы SQLServerAgent;
– Log – сообщения об ошибках;
– Repldata – для тиражирования.

Основным инструментом администрирования и создания объектов базы данных является программа Management Studio.

6.2. Администрирование MS SQL сервера

Администратор сервера баз данных выполняет следующие функции:

– учет и управление разработанными объектами: структурами данных и программными объектами;

– управление безопасностью:

- управление полномочиями пользователей;
- мониторинг использования баз данных и сервера на основе системных и специальных журналов;

– обеспечение надежности: разработка стратегии и настройка резервного копирования;

– мониторинг и выявление проблем производительности, разработка методов измерения производительности, оптимизация настроек серверов, разработка сценариев;

– восстановление данных после сбоев.

При наличии соответствующих прав Management Studio позволяет выполнять создание базы данных, таблиц, представлений пользователя, процедур, триггеров и других объектов базы данных, управлять подключениями и правами пользователей, получать информацию о текущей работе сервера – словом, выполнять все работы по администрированию сервера, используя привычный для пользователей Windows графический интерфейс (рис. 18).

Для выполнения большинства функций достаточно на левой панели указать группу объектов или объект и правой кнопкой мыши вызвать контекстное меню. Группа объектов Databases содержит список всех баз данных сервера. Вкладка Security позволяет управлять полномочиями на уровне сервера (систему безопасности базы данных обеспечивается отдельно аналогичной вкладкой внутри базы данных. Security позволяет просматривать и управлять подключением к серверу пользователей и других серверов. Management отображает компоненты управления сервером.

6.3. Система безопасности

Подключение к SQL-серверу выполняется по регистрационному имени (login) пользователя в сети. Это сетевое имя должно быть зарегистрировано на SQL-сервере. При этом возможна аутентификация независимая от Windows NT или интегрированная с Windows NT. Второй вариант удобней в администрировании, так как сетевое имя и пароль запрашиваются и проверяются контроллером домена при регистрации пользователя, и при успешной аутентификации пользователь получает определенные права на SQL-сервере.

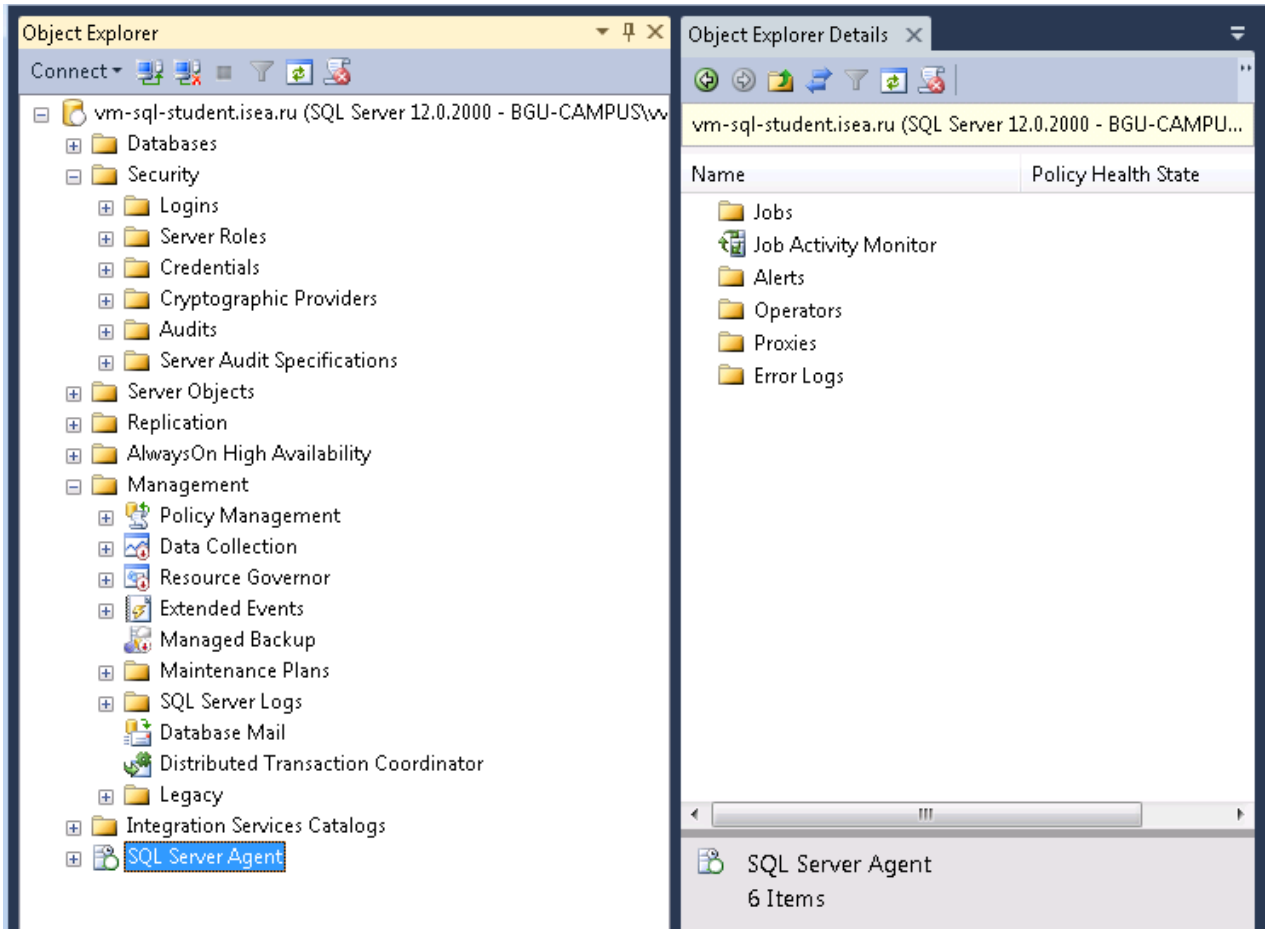


Рис. 18. Окно программы Management Studio

Права на уровне сервера присваиваются пользователю сети включением его в одну из стандартных ролей сервера (табл. 2). Кроме этого, пользователь сети может получить определенные права в каждой базе данных сервера.

Таблица 2

Стандартные роли на уровне сервера

Встроенная роль сервера	Назначение
Sysadmin	Может выполнять любые действия в SQL-сервере
Serveradmin	Выполняет конфигурирование и выключение сервера
Setupadmin	Управляет связанными серверами и процедурами, автоматически запускающимися при запуске SQL-сервера
Securityadmin	Управляет учетными записями и правами на создание базы данных, также может читать журнал ошибок
Processadmin	Управляет процессами, запущенными в SQL-сервере
Dbcreator	Может создавать и модифицировать базы данных
Diskadmin	Управляет файлами SQL-сервера

Каждая база данных имеет свой список пользователей БД – user-ов (не путать с пользователями сети – login-ами). Пользователь сети (login) может быть отображен в определенного пользователя БД (user-а). Один пользователь БД создается в момент создания БД и не может быть удален – это владелец БД (dbo). Он обладает всеми полномочиями в БД и раздает права всем остальным

пользователям. Системный администратор SQL-сервера (sa, созданный в момент инсталляции, или login, включенный в роль Sysadmin) отображается во владельца (dbo) в каждой базе сервера.

Для удобства администрирования в БД вводятся роли (группы пользователей). Права и запреты на операции с объектами БД обычно раздаются ролям, хотя могут назначаться и непосредственно пользователям. В дополнении к персональным разрешениям и запретам пользователь получает разрешения и запреты ролей, в которые он включен. В случае противоречия разрешений и запретов последние имеют приоритет. Если пользователь персонально или благодаря включению в некоторую роль получает право выполнять команду, а другая его роль содержит запрет на выполнение этой команды, то в результате пользователю будет запрещено выполнять эту команду. Итак, чтобы выполнить команду, пользователь должен получить разрешение и не иметь запретов на ее выполнение.

Предусмотрены стандартные роли базы данных (табл. 3).

Таблица 3

Стандартные роли пользователя базы данных

Встроенная роль базы данных	Назначение
db_owner	Имеет все права в базе данных
db_accessadmin	Может добавлять или удалять пользователей
db_securityadmin	Управляет всеми разрешениями, объектами, ролями и членами ролей
db_ddladmin	Может выполнять любые команды DDL, кроме GRANT, DENY и REVOKE
db_backupoperator	Может выполнять команды DBCC, CHECKPOINT и BACKUP
db_datareader	Может просматривать любые данные в любой таблице БД
db_datawriter	Может модифицировать любые данные в любой таблице БД
db_denydatareader	Запрещается просматривать данные в любой таблице
db_denydatawriter	Запрещается модифицировать данные в любой таблице

Для создания или управления подключениями пользователей к серверу используется список Security\Logins. Как правило, пользователь не получает полномочий на уровне сервера (не включается в стандартные роли на уровне сервера). Для него определяют базы данных, с которыми он будет работать при выполнении служебных обязанностей, и отображают в определенного пользователя (user-a) в каждой базе данных. На Рис. 19 login пользователя BGU-CAMPUS\0154609 отображается в пользователя «менеджер» базы «Торговля», включенного в роли «Менеджеры» и «Public» этой базы.

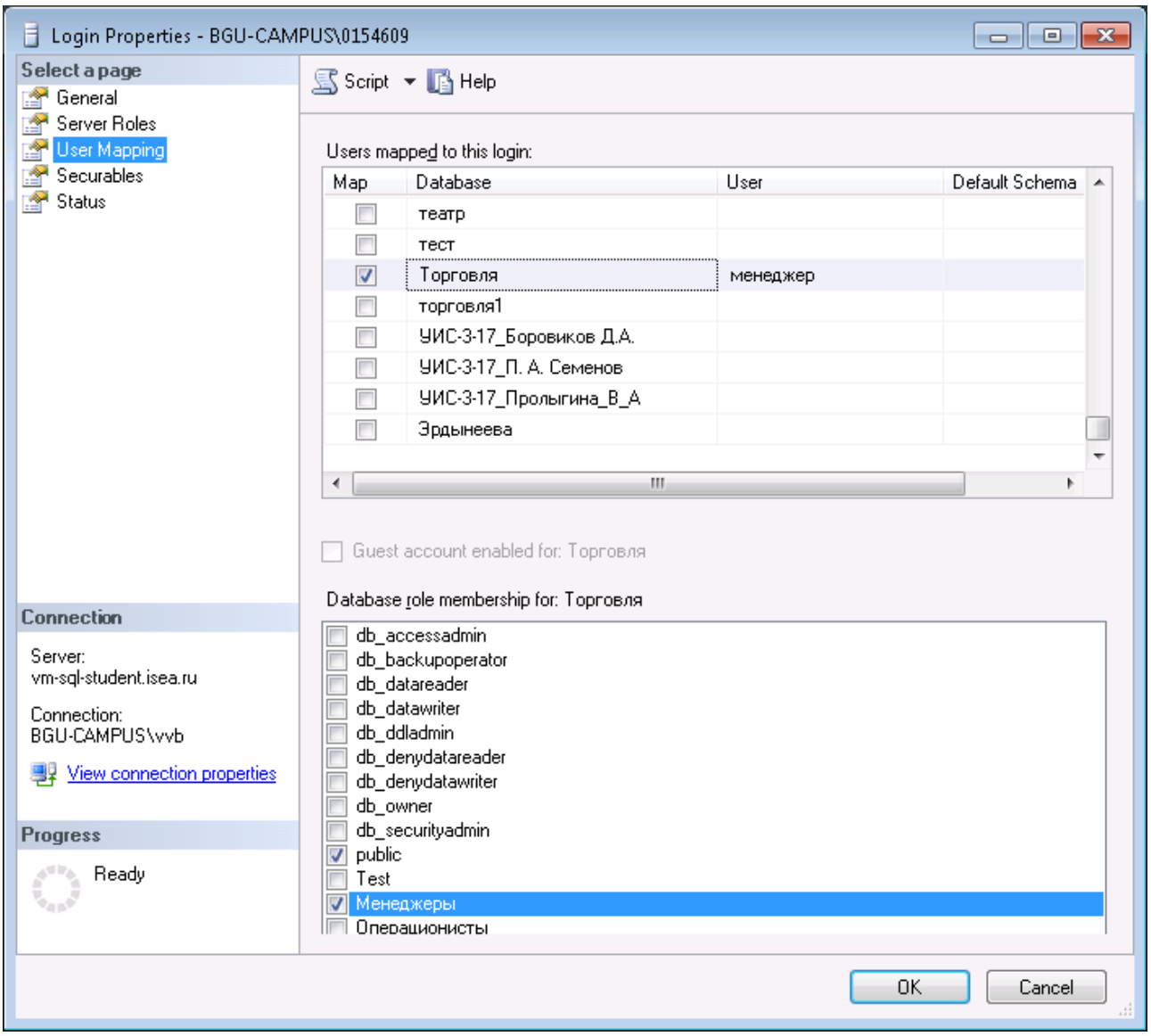


Рис. 19. Параметры (свойства) подключения (login-а) к серверу

Права и запреты определяются для объектов БД (таблицы, представления, процедуры, табличные функции) и соответствующих операций SELECT, INSERT, UPDATE, DELETE, EXECUTE. Полномочия предоставляются командой Manage permissions в контекстном меню объекта, пользователя или группы (рис. 20). Для операций SELECT и UPDATE можно определить разрешения и запреты на уровне колонок (кнопка Columns Permissions). В дополнении к персональным разрешениям и запретам пользователь получает разрешения и запреты ролей, в которые он включен. Конфликт разрешения и запрета на выполнение какой-либо операции решается в пользу запрета.

Кроме перечисленных, предусмотрены разрешения на специальные операции, такие как создание объектов БД. Эти права прописываются в свойствах базы данных.

Полномочия имеет право предоставлять владелец (создатель) объекта (таблицы, представления, табличной функции).

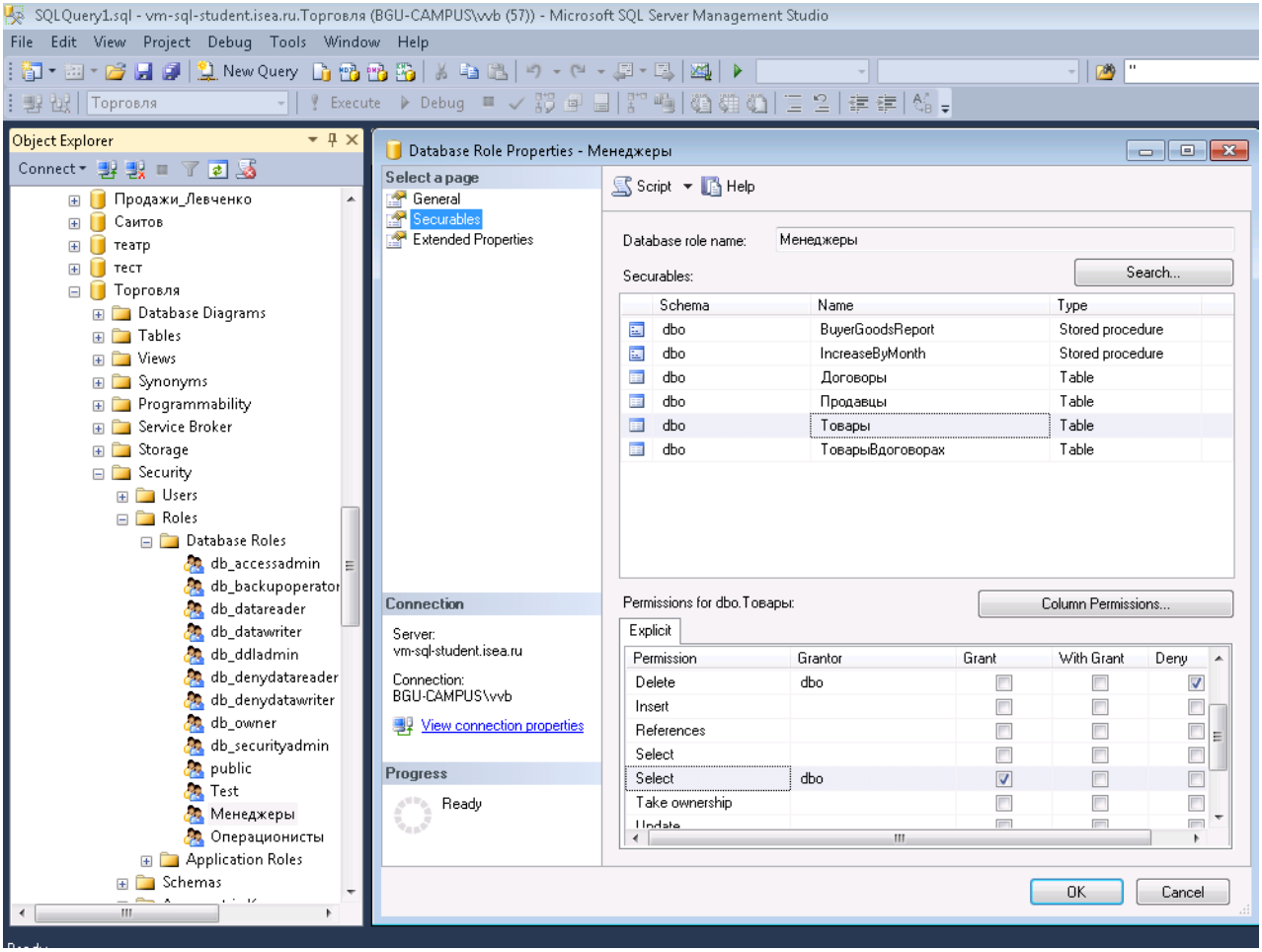


Рис. 20. Форма определения прав выполнения операций с объектами БД

6.4. Резервное копирование и восстановление БД

Резервное копирование баз данных является необходимым компонентом системы защиты данных. Обычно резервное копирование выполняется периодически и предусматривает копирование всей БД и (или) журнала транзакций во время минимальной загруженности сервера. Например, еженедельное копирование БД и ежедневное копирование журнала. В случае жесткого сбоя, повлекшего потерю содержания основного носителя БД, данные восстанавливаются на момент последнего резервного копирования.

Копирование журнала транзакций позволяет восстанавливать изменения БД по журналу и копии БД не только на момент копирования, но также на любой момент времени (переключатель Timeline… на рис. 22). Кроме этого, копия журнала транзакций более компактна, и ее создание требует меньше ресурсов.

Копирование можно запустить командой Backup в контекстном меню базы данных в программе Management Studio (рис. 21). Обычно резервное копирование выполняют автоматически при помощи специального задания для MS SQL Agent. Создается такое задание при помощи мастера резервного копирования (Backup Wizard) или специальной формы. Основным параметром на автоматическое резервное копирование является время запуска копирования. Напомним, что такие задания выполняются службой SQLServerAgent, которая для этого должна быть запущена.

Резервное копирование кроме создания копии БД приводит к очистке журнала транзакций.

Кроме копирования всей БД и журнала транзакций MS SQL сервер предусматривает разностное (Differential) копирование – копирование страниц, измененных с момента последнего копирования, – и копирование выбранных групп файлов.

Восстановление базы данных может быть выполнено командой Restore. Параметры этой команды представлены на рис. 22.

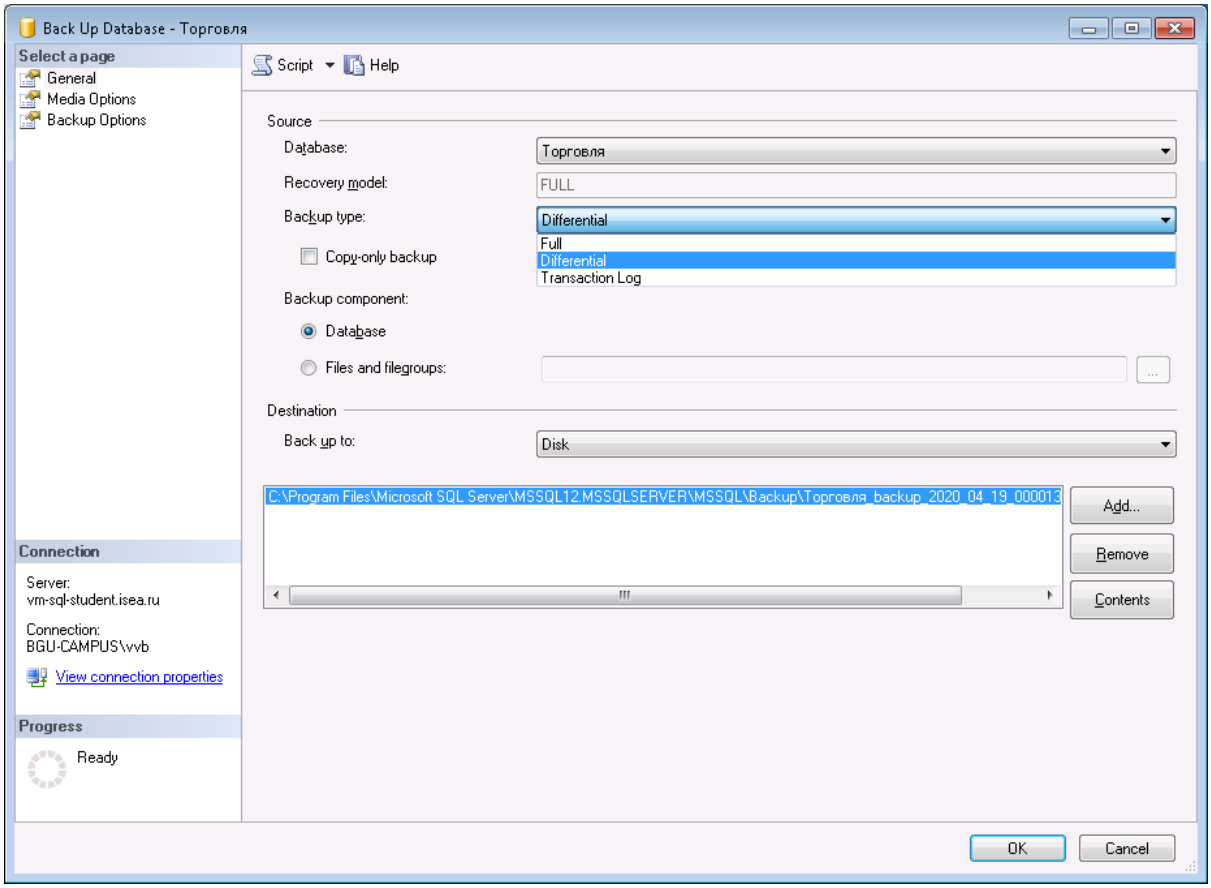


Рис. 21. Определение параметров команды копирования
в Management Studio

6.5. Возможности контроля объектов базы данных

Многие возможности администрирования можно использовать в программных объектах БД в виде специальных команд, системных процедур и функций. Например, команда Grant определяет права пользователя. Системные процедуры начинаются с префикса SP_ и предназначены для выполнения действий по настройке системы и получения свойств и параметров объектов базы данных. Многие процедуры являются командным вариантом выполнения функций администрирования. Например, команда sp_adduser добавляет нового пользователя БД.

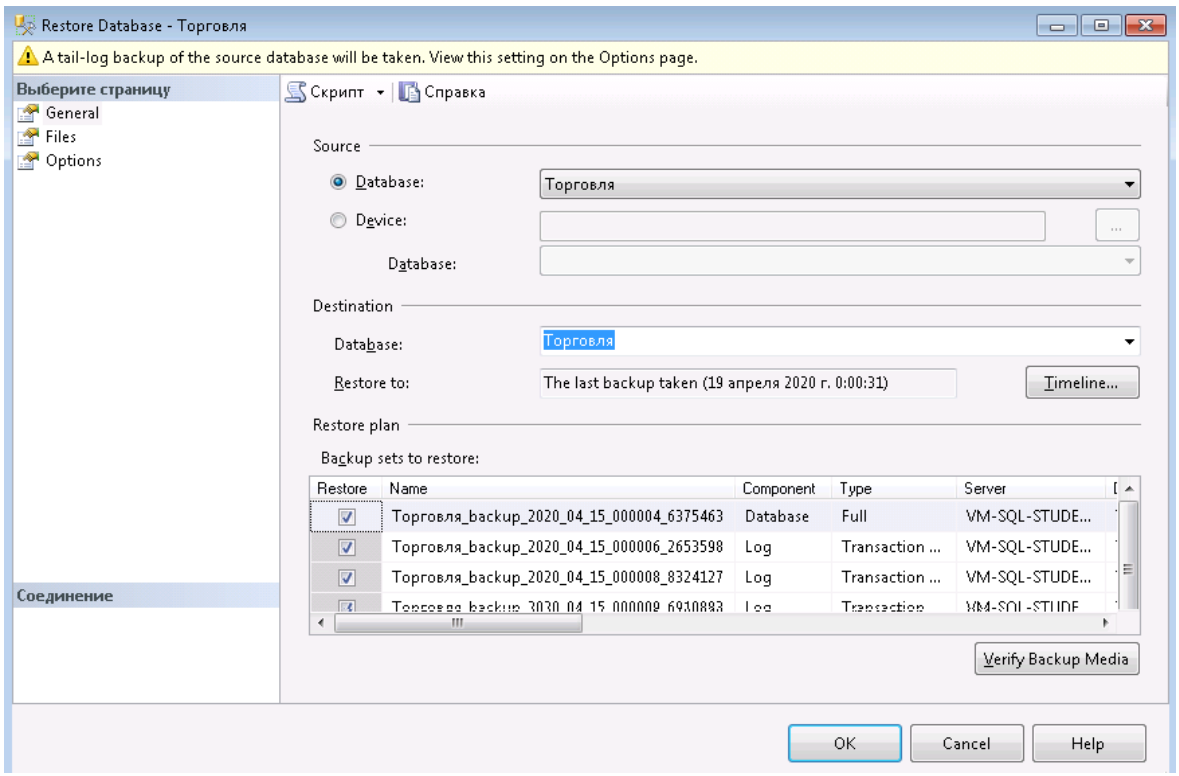


Рис. 22. Определение параметров команды восстановления в Management Studio

Ниже перечислены системные функции по определению пользователей сети и базы данных:

– *UserName()* – имя пользователя БД;

– *IS_MEMBER ('<имя роли>')* – проверяет принадлежность пользователя указанной роли;

– *SUSER_SID (['<логин>'])* возвращает уникальный sid (security identification number) пользователя сети;

– *SUSER_SNAME ([<sid пользователя>])* возвращает логин пользователя.

Эти функции можно применять в процедурах, функциях, триггерах для определения, кто выполняет соответствующие действия, и на этой основе выполнять более сложные правила предоставления полномочий. Например, ограничивать по времени выполнение некоторых операций определенным ролям и пользователям.

Для получения статистики выполнения запросов можно использовать дополнительные возможности. Кнопка Display Estimated Execution Plan открывает дополнительную панель Execution Plan (Рис. 23), на которой демонстрирует последовательность действий по выполнению запроса. Это бывает полезно для оптимизации выполнения запросов.

Кнопка Include Client Statistics включает соответствующую панель (Рис. 24), которая содержит данные по взаимодействию клиента и сервера. Статистика позволяет выделить наиболее ресурсоемкие части запросов.

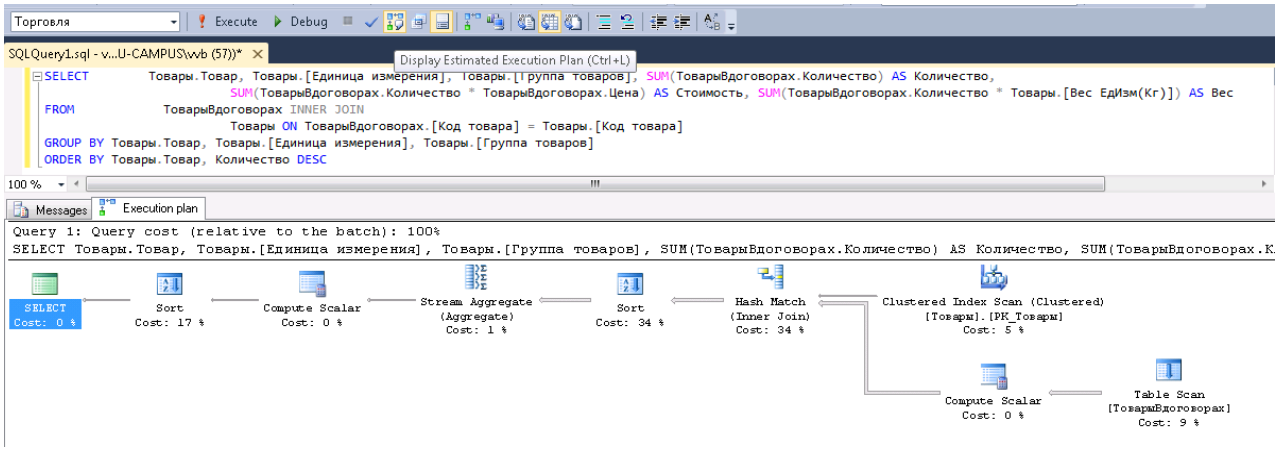


Рис. 23. Панель Execution Plan

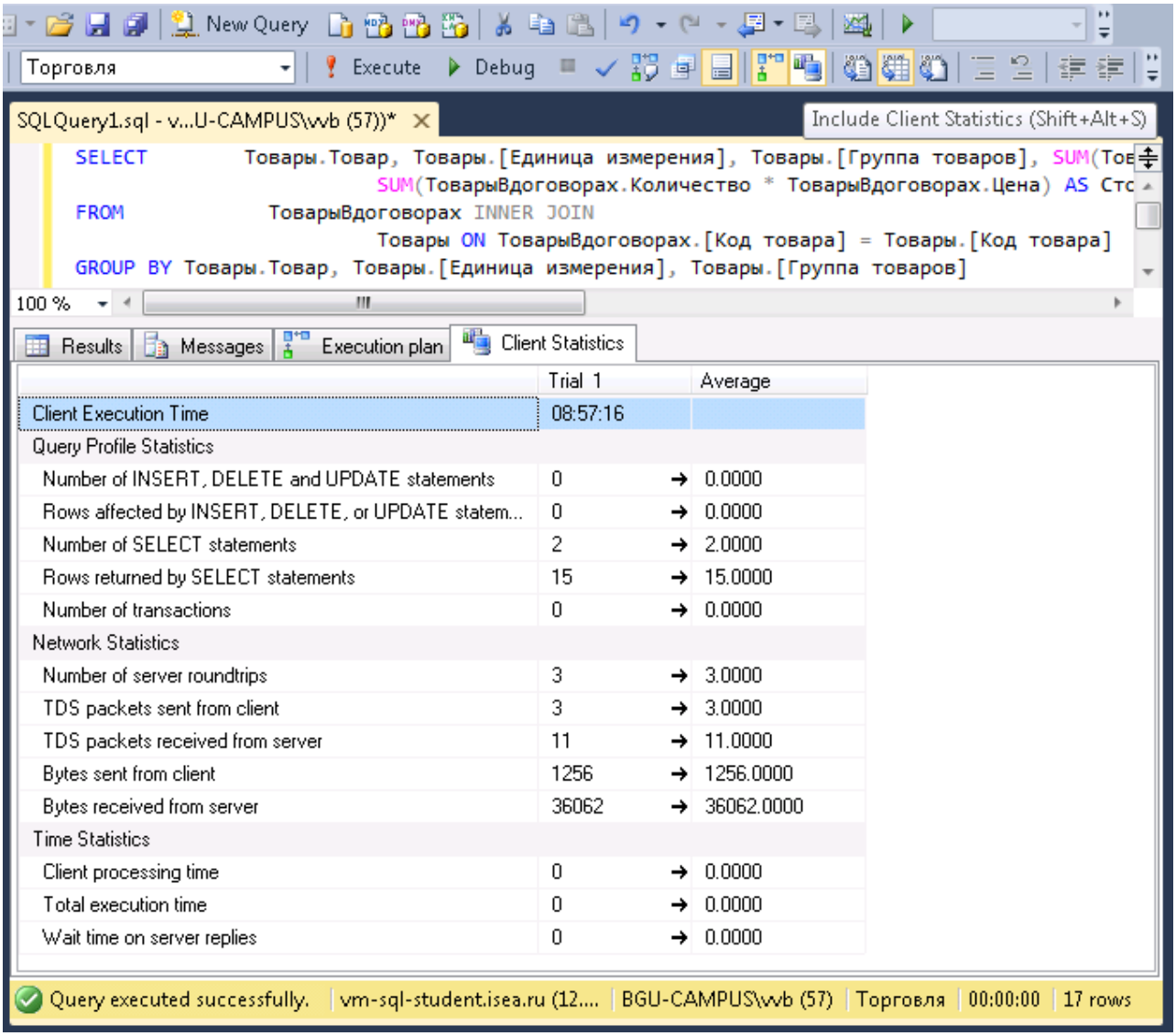


Рис. 24. Панель Client Statistics

Наблюдение взаимодействия клиента и сервера можно выполнить при помощи программы SQL Profiler. Для этого надо входить в группу администраторов сервера. Программа SQL Profiler обеспечивает выполнение следующих работ:

– контроль работы SQL-сервера;

– отладка SQL-команд и -процедур;

– аудит работы SQL-сервера.

Обычно настройка трассировки в SQL Profiler заключается в следующем:

– запускается SQL Profiler;

– стартует новая трассировка. При необходимости указывается запись трассировки в файл;

– на вкладке Event выбирают события, отображаемые в трассировке;

– на вкладке Data Columns выбирают поля, описывающие события;

– на вкладке Filters устанавливают фильтрацию событий (например, по имени базы данных, логину пользователя и т.п.).

Вопросы

1. Перечислите и укажите назначение сетевых служб MS SQL сервера.
2. Перечислите и укажите назначение системных баз данных MS SQL сервера.
3. Перечислите и укажите назначение основных утилит MS SQL сервера.
4. Опишите систему безопасности MS SQL сервера. Как происходит аутентификация?
5. Кто и каким образом получает полномочия на уровне сервера? Какие это полномочия?
6. Кто и каким образом получает права на уровне базы данных? Какие это права?
7. Как соотносятся понятия «пользователь сети», «группа сети», «пользователь БД», «роль БД»?
8. Опишите встроенные роли и пользователей БД.
9. Какие права, для каких объектов могут получать пользователи?
10. Как определяются права и запреты пользователя, если он является членом нескольких ролей и обладает персональными правами и запретами?
11. Перечислите виды резервного копирования, укажите их назначение и условия их применения.
12. Перечислите преимущества резервного копирования журнала транзакций.
13. Как можно получить сведения об особенностях и статистике выполнения запросов?
14. Как получить сведения о взаимодействии пользователей с SQL-сервером?

7. Многомерный анализ данных

7.1. Структура многомерных данных

Анализ показателей, изучение зависимости показателей от признаков (факторов), исследование взаимного влияния показателей является основой аналитической работы. Прежде чем формулировать зависимости в аналитическом виде, необходимо эти зависимости «увидеть». Для этого изучают таблицы, отчеты, графики, демонстрирующие зависимости показателей и признаков.

Компьютерные технологии позволяют оперативно выполнять такую обработку по запросу аналитика. Для этого была предложена технология многомерного анализа данных – On-Line Analytical Processing (OLAP) [6; 7]. OLAP использует специальные структуры – хранилища данных (Data warehouses) или многомерные базы данных для обеспечения хранения, агрегирования и извлечения данных. Эта технология качественно превосходит формирование и печать отчетов: результат выдается немедленно, в удобной форме с соответствующими графиками и диаграммами. Пользователь может выбрать любое направление (комбинацию признаков) для агрегирования или детализации. Результатом многомерного анализа являются выявление некоторых феноменов: отклонений, выбросов или провалов, тенденций, зависимостей.

Данные в хранилище попадают из систем регистрации данных (Online Transaction Processing – OLTP), которые предназначены для автоматизации бизнес-процессов и решают в частности учетные задачи. В таких системах регистрируются все значимые для управления события: продажи, отгрузки, платежи и т.д. Хранилище может пополняться не только сведениями из баз данных, но и за счет внешних источников, например, различных статистических отчетов, биржевых котировок, курсов валют. Данные в разных источниках могут иметь разные структуры, разные кодировки и форматы. Поэтому в процессе загрузки данных в хранилище приходится решать задачи по преобразованию систем классификации и форматов данных.

Технология OLTP позволяет выполнять вычисления, определять значения показателей для разных комбинаций признаков. Тем не менее целесообразно применять OLAP-технологию по следующим причинам:

1. Необходим специальный инструментарий и эффективная реализация вычислений значений показателей для произвольного набора признаков и выявления зависимостей.

2. Анализировать данные оперативных систем учета напрямую невозможно или очень затруднительно: данные хранятся разрозненно в разных подсистемах, в разных СУБД, в форматах, на разных серверах корпоративной сети. Аналитическую обработку затрудняют сложность и запутанность структур хранения данных, избыточность детальной информации.

3. Сложные аналитические запросы к оперативной информации тормозят текущую работу информационной системы, надолго блокируя таблицы и захватывая ресурсы сервера.

К хранилищам данных предъявляют требования, сведенные в тест FASMI – Fast Analysis of Shared Multidimensional Information, предложенный Найджелом Пендсом (Nigel Pendse):

1. Fast (Быстрый) – анализ должен производиться одинаково быстро по всем аспектам информации. Приемлемое время отклика – 5 секунд или менее.

2. Analysis (Анализ) – должна быть возможность осуществлять основные типы числового и статистического анализа.

3. Shared (Разделяемой) – много пользователей должны иметь доступ к данным, при этом необходимо контролировать доступ к конфиденциальной информации.

4. Multidimensional (Многомерной) – показатели должны вычисляться для произвольного набора классификационных признаков.

5. Information (Информации) – приложение должно иметь возможность обращаться к любой нужной информации, независимо от ее объема и места хранения.

В OLAP пользователь получает естественную, интуитивно понятную модель данных, представленную в виде многомерных ***кубов*** (Cubes). ***Измерениями*** (Dimensions) многомерной системы координат куба служат атрибуты (признаки) анализируемого бизнес-процесса. ***Измерение*** – набор значений для идентификации некоторого свойства бизнес-процесса. Значения, «откладываемые» вдоль измерений, называются ***метками*** (members). Например, для анализа продаж измерениями могут быть товар, регион, тип покупателя, дата. Метками измерения «Товар» будут все наименования товаров из конкретной предметной области.

Измерение может иметь иерархическую структуру – задавать некоторую классификацию значений. Измерение «Товар» может включать классификацию, например, с использованием общероссийского классификатора продукции по видам экономической деятельности (ОКПД). В этом случае измерение описывается не одним, а несколькими атрибутами – по числу уровней иерархии в классификации. Стандартную иерархию образует время, для которого вводится набор из общеизвестных уровней (…, час, день, неделя, месяц, квартал, год, …). Кроме этого, с каждой меткой некоторого измерения могут быть связаны дополнительные свойства. Например, для даты можно определить свойство «день недели» (понедельник, вторник, …).

Фиксированные значения всех измерений задают ***ячейку куба*** (cell), с которой связан набор ***показателей*** или ***мер*** (Measures), количественно характеризующих процесс. Это могут быть объемы продаж в штуках или в денежном выражении, остатки на складе, издержки и т.п.

В качестве показателей в кубе, изображенном на Рис. 25, использованы стоимости (верхнее число) и количество (нижнее число) проданных единиц, а в качестве измерений – продавцы, товары и покупатели.

Для каждого показателя определяется правило его вычисления. Фиксация меток измерений определяет множество данных, по которым вычисляется показатель. При этом применяются различные функции агрегирования. Наиболее распространенной функцией агрегирования является сумма. Кроме суммы,

применяются вычисление количества, минимума, максимума, среднего и других статистических характеристик.

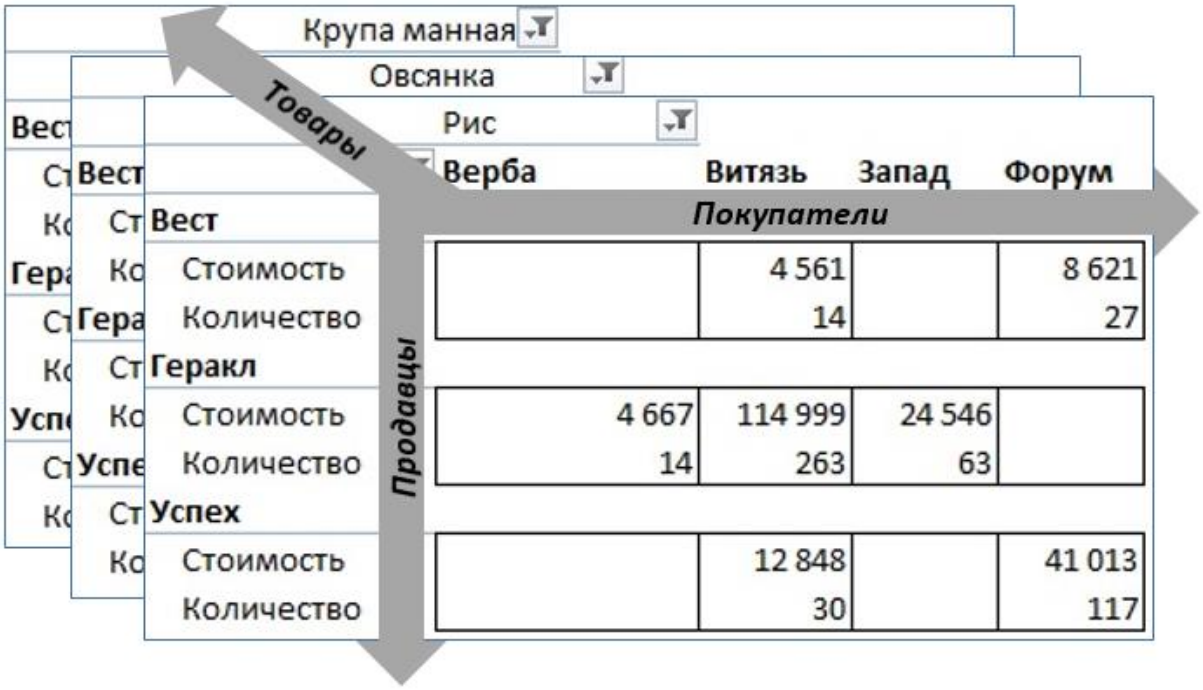


Рис. 25. Пример куба

Некоторые показатели вычисляются на основе других показателей. Например, средняя цена вычисляется как суммарная стоимость продаж, деленная на суммарное количество проданного товара. По способу вычисления показатели можно разделить на две категории: агрегированные показатели, определенные с помощью функций агрегирования по исходным данным, и вычисляемые показатели, определенные по другим показателям.

Набор показателей фактически рассматривается как одно дополнительное измерение (Measures): его метками являются названия показателей.

7.2. Операции над многомерными данными

Для многомерных данных вводят специальные операции исследования показателей в зависимости от выбранных комбинаций измерений. Операции над кубами снова дают куб. Одна ячейка – это тоже куб, только без измерений.

Формирование среза. ***Срез*** (Slice) – это часть куба, получившаяся в результате фиксации значения одного или более измерений. Результатом среза снова является куб с меньшим количеством ячеек. Если выбирается метка, соответствующая некоторому уровню иерархии, то в результате уменьшается количество меток измерения и, соответственно, ячеек. Например, при выборе категории «Напитки» в измерении «Товары» остаются метки различных напитков. При выборе метки конкретного товара измерение «Товары» свертывается до точки и количество измерений в срезе куба уменьшается.

Если фиксируются значения всех измерений, кроме двух, получается обычная двумерная таблица. В этом случае горизонтальная ось (заголовки колонок таблицы) представлена метками одного измерения, в вертикальная ось (заго-

ловки строк) – метками другого, а в ячейках таблицы размещаются значения показателей. На Рис. 26 изображен двумерный срез куба – зафиксировано значение «Крупа манная». Осталось два измерения – «Покупатели» и «Продавцы».

Товар = «Крупа манная»		Покупатели			
Продавцы		Верба	Витязь	Запад	Форум
Вест	Стоимость	5 244	33 941	5 167	
	Количество	17	67	20	
Геракл	Стоимость	53 683	35 294		
	Количество	191	69		
Успех	Стоимость				87 188
	Количество				181

Рис. 26. Двумерный срез куба для показателя «стоимость»

Операция агрегирования (Drill Up) – переход от детализированных данных к агрегированным. Агрегирование выполняется при уменьшении числа измерений или при переходе к обобщающим уровням иерархии. При этом для каждого показателя куба по множеству детальных значений нужно вычислить агрегированное значение показателя. Например, агрегирование по измерению «Товар» приведет к получению куба на Рис. 27, аналогичного представленному на Рис. 26. Разница будет в значениях показателей, которые для агрегирования будут вычисляться по всем товарам, а не только для товара «Крупа манная». Стоимость и количество для среза выбираются из ячеек, а для агрегирования вычисляются суммированием стоимостей и количеств продаж для продавца и покупателя.

Товар = «Все»		Покупатель			
Продавцы	Значения	Верба	Витязь	Запад	Форум
Вест	Стоимость	108 038	221 158	145 868	220 280
	Количество	609	1014	495	1514
Геракл	Стоимость	220 082	172 823	41 417	
	Количество	1269	479	167	
Успех	Стоимость	56 047	84 158		195 465
	Количество	369	660		782

Рис. 27. Агрегирование по товарам

Операция агрегирования приводит к уменьшению размерности, если она выполняется для всего измерения. В примере на Рис. 27 в результате агрегирования исчезает измерение «Товары». Если агрегирование связано с переходом к обобщающим уровням иерархии, то соответствующее измерение не исчезает. При агрегировании данных по месяцам будут получены обобщающие значения показателей в каждом месяце, однако временное измерение останется. В нем вместо трех уровней иерархии – год, месяц, день – останется два: год и месяц.

Операция детализации (Drill Down) заключается в переходе от агрегированных к более детализированным данным. Детализация появляется при до-

бавлении ранее свернутого измерения – при этом уровень агрегирования понижается: каждая ячейка куба заменяется набором ячеек по количеству меток в добавляемом измерении, а каждое значение показателя рассчитывается для новой комбинации координат измерений. На Рис. 28 представлена замена агрегированных значений показателей детальными.

Детализация может быть связана с появлением свернутого измерения (на Рис. 28) появляется измерение «Товары») и увеличением размерности. Детализация также может быть вызвана переходом на более детальный уровень измерения без изменения количества измерений.

			Покупатель			
Продавцы	**Товар**	**Значения**	**Верба**	**Витязь**	**Запад**	**Форум**
Вест	**Крупа манная**	Стоимость	5 244	33 941	5 167	
		Количество	17	67	20	
	Овсянка	Стоимость	3 840	50 565	2 778	69 950
		Количество	8	119	6	196
	Рис	Стоимость		4 561		8 621
		Количество		14		27

Рис. 28. Детализация агрегированных значений показателей

Работа пользователя с кубом обычно включает все перечисленные операции. К ним также относят перестановки осей в процессе визуализации. Первоначально менеджера интересуют агрегированные значения показателей, скажем сумма всех продаж за последний период. Затем можно выполнить детализацию этого показателя по категориям товаров или товарам для определения товаров, имеющих максимальную стоимость продаж. Возможно выполнить сечение по предыдущему периоду для анализа тенденций в изменении суммы продаж в целом и по отдельным категориям. Невозможно заранее предугадать направление анализа, тем не менее перечисленные операции позволяют реализовать любое из них.

7.3. Архитектура OLAP-средств

Функции обработки многомерных данных в OLAP-приложениях может быть разделена на три уровня:

1. Многомерное представление данных – средства конечного пользователя, обеспечивающие многомерную визуализацию и задание команд многомерного анализа (срез, агрегирование, детализация) и управление отображением получаемых данных.

2. Многомерная обработка – средство (язык) формулирования многомерных запросов (традиционный реляционный язык SQL здесь оказывается непригодным) и процессор, умеющий обработать и выполнить такой запрос.

3. Многомерное хранение – средства физической организации данных, обеспечивающие эффективное выполнение многомерных запросов и управление полномочиями доступа к многомерным данным.

Конкретные OLAP-продукты, как правило, представляют собой либо средство многомерного представления данных – OLAP-клиент (например, сводные таблицы в Excel или ProClarity фирмы Knosys), либо многомерную серверную СУБД – OLAP-сервер (например, Oracle Express Сервер или Microsoft Analysis Services).

Слой многомерной обработки обычно бывает встроен в OLAP-клиент и (или) в OLAP-сервер, но может быть выделен в чистом виде, как, например, компонент Pivot Table Service фирмы Microsoft.

В решениях фирмы Microsoft основным компонентом аналитических служб является Analysis Services. Этот компонент предназначен для создания OLAP-кубов на основе реляционных хранилищ данных, а также для предоставления доступа к ним из клиентских приложений.

Теоретически OLAP-куб, созданный с помощью аналитических служб Microsoft, может содержать все данные из разных источников, хранимые в ячейках кубов, плюс агрегированные значения, которые соответствуют уровням иерархии измерений и различным комбинациям меток измерений. Аналитические службы могут производить динамическое обновление куба, если в исходные таблицы были добавлены новые записи.

Аналитические службы сохраняют агрегированные данные только для функций агрегирования (суммы, количества экземпляров, максимальные и минимальные значения и др.). Однако в случае необходимости можно создавать так называемые вычисляемые члены (calculated members) для получения других типов агрегированных значений (средних, средневзвешенных, дисперсий и т.д.). При этом, помимо применения встроенных средств создания агрегированных данных, Analysis Services позволяют использовать для вычисления агрегированных данных функции VBA или Excel, а также создавать собственные средства.

Наконец, аналитические службы Microsoft позволяют создавать так называемые виртуальные кубы (virtual cubes), которые в определенной степени являются аналогами представлений (view) реляционных СУБД. Виртуальные кубы не содержат данных, но позволяют представить в виде единого куба данные из нескольких кубов, имеющих хотя бы одно общее коллективное измерение.

Decision Support Objects (DSO) – это набор библиотек, содержащих COM-объекты, позволяющие создавать и модифицировать многомерные базы данных и содержащиеся в них объекты (кубы, коллективные измерения и т.д.). Эти библиотеки можно использовать для разработки собственных приложений, в которых осуществляется создание или модификация многомерных баз данных, в том числе и для реализации действий, не предусмотренных в составе аналитических служб.

Приложения в операционной системе MS Windows, предназначенные для чтения OLAP-данных, при взаимодействии с аналитическими службами обязательно используют PivotTable Service – библиотеки, загружаемые в адресное пространство клиентского приложения. Эти библиотеки автоматически устанавливаются вместе с аналитическими службами (независимо от того, какая именно их часть установлена – клиентская или серверная), а также вместе с Mi-

crosoft Office. В состав Microsoft SQL Server входит также инсталляционное приложение для установки PivotTable Service на компьютер, на котором не установлены эти службы.

Для взаимодействия с PivotTable Service клиентское приложение может использовать OLE DB for OLAP – расширение универсального механизма доступа к данным OLE DB, позволяющее обращаться к многомерным данным, а также ADO MD – библиотеки, представляющие собой надстройку над OLE DB for OLAP и являющиеся COM-серверами для доступа к многомерным данным, удобными для применения в клиентских приложениях.

Отметим, что спецификация OLE DB for OLAP является открытой. Это означает, что можно создавать и другие OLAP-серверы, поддерживающие OLE DB for OLAP (либо разрабатывать OLE DB-провайдеры к уже имеющимся OLAP-средствам), а также создавать клиентские приложения, обращающиеся к любым таким источникам данных с помощью PivotTable Service, OLE DB for OLAP и ADO MD.

Из других клиентских приложений, не входящих в состав аналитических служб, но часто используемых для просмотра OLAP-кубов, следует назвать приложения Microsoft Office, в частности Microsoft Excel. С помощью Excel можно обращаться к серверным OLAP-кубам, получая их двух- и трехмерные сечения или проекции на листах рабочих книг Excel в виде сводных таблиц, а также создавать локальные OLAP-кубы в виде файлов на основе реляционных данных, доступных с помощью OLE DB.

Кроме того, в состав Microsoft Office Web Components входит элемент управления ActiveX PivotTable List, позволяющий реализовать сходную функциональность как в обычном Windows-приложении, так и на HTML-странице, предназначенной для применения внутри корпоративной сети.

7.4. Создание многомерных баз данных

Программа Management Studio позволяет выполнять все функции по администрированию многомерных баз данных. Для создания хранилища следует подключиться к аналитическим службам MS SQL сервера, выбрав команду «Соединить» \ «Службы Analysis Services» окне обозревателя объектов и указав имя сервера. После этого (при наличии полномочий) можно выполнять все необходимые команды для администрирования и доступа к данным. Реализация OLAP-технологии начинается с создания многомерной базы данных. Это можно сделать, выбрав команду «Создать базу данных» в контекстном меню списка многомерных баз данных. Для многомерной базы определяется имя базы данных и другие ее параметры. Далее в самой базе определяются источники данных для кубов, измерения и сами кубы. Для этого используется SQL Server Data Tools (SSDT), включенные в среду разработки Microsoft Visual Studio.

В среде Microsoft Visual Studio создается проект командой «Создать» \ «Проект» с последующим выбором среди шаблонов Business Intelligence «Проект многомерных данных и интеллектуального анализа» с заданием имени и расположения файлов проекта (Рис. 29).

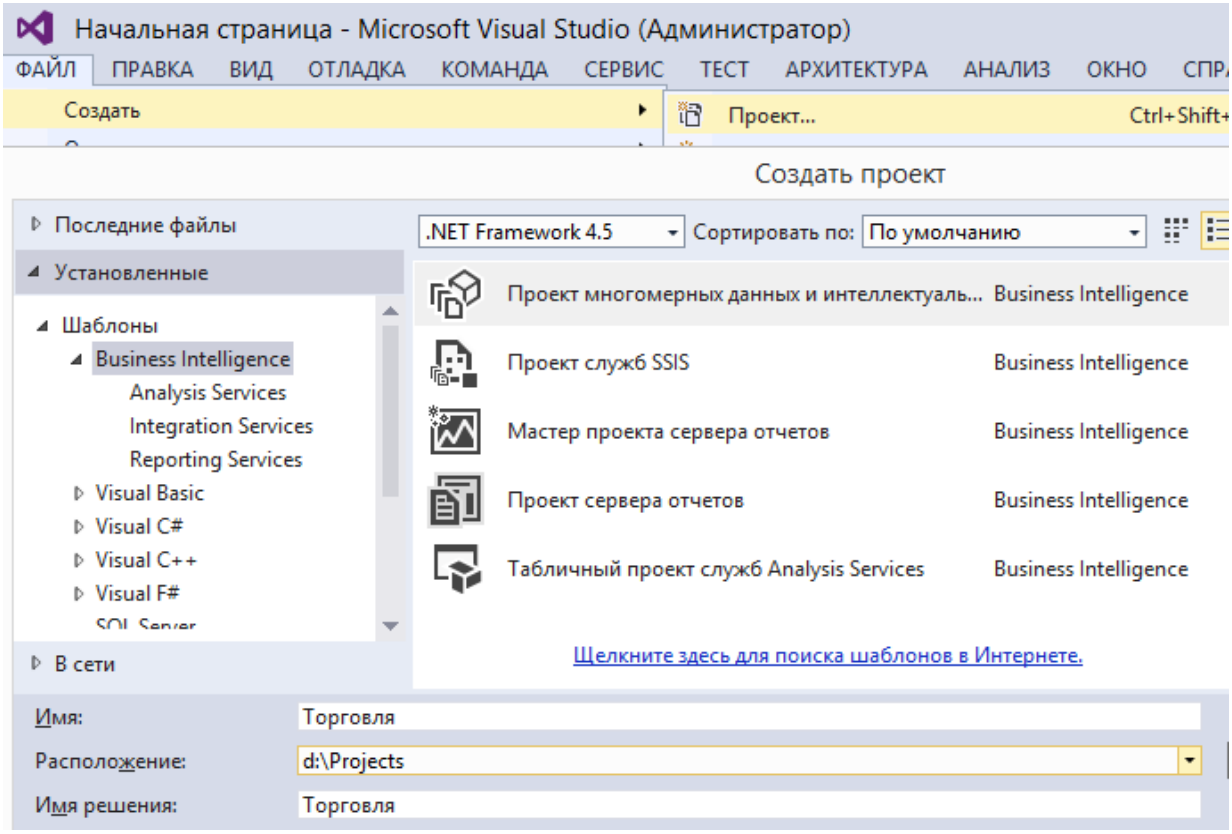


Рис. 29. Создание проекта многомерной базы данных

Далее для определения компонентов будет использоваться обозреватель решений (Рис. 30) и команды контекстного меню.

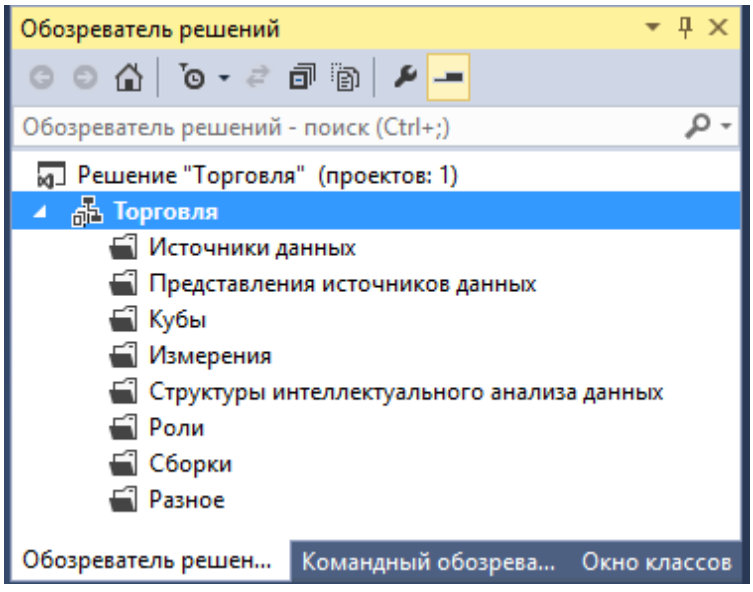


Рис. 30. Обозреватель решений проектов

Одно из основных свойств проекта (свойства проекта открывает одноименная команда контекстного меню) – это параметры развертывания (Рис. 31): имя сервера и многомерная база данных. После определения проекта все компоненты его будут развернуты в указанной многомерной базе данных. Отделение проекта и структур многомерной базы данных позволяет неоднократно развертывать многомерные базы данных в различных серверах.

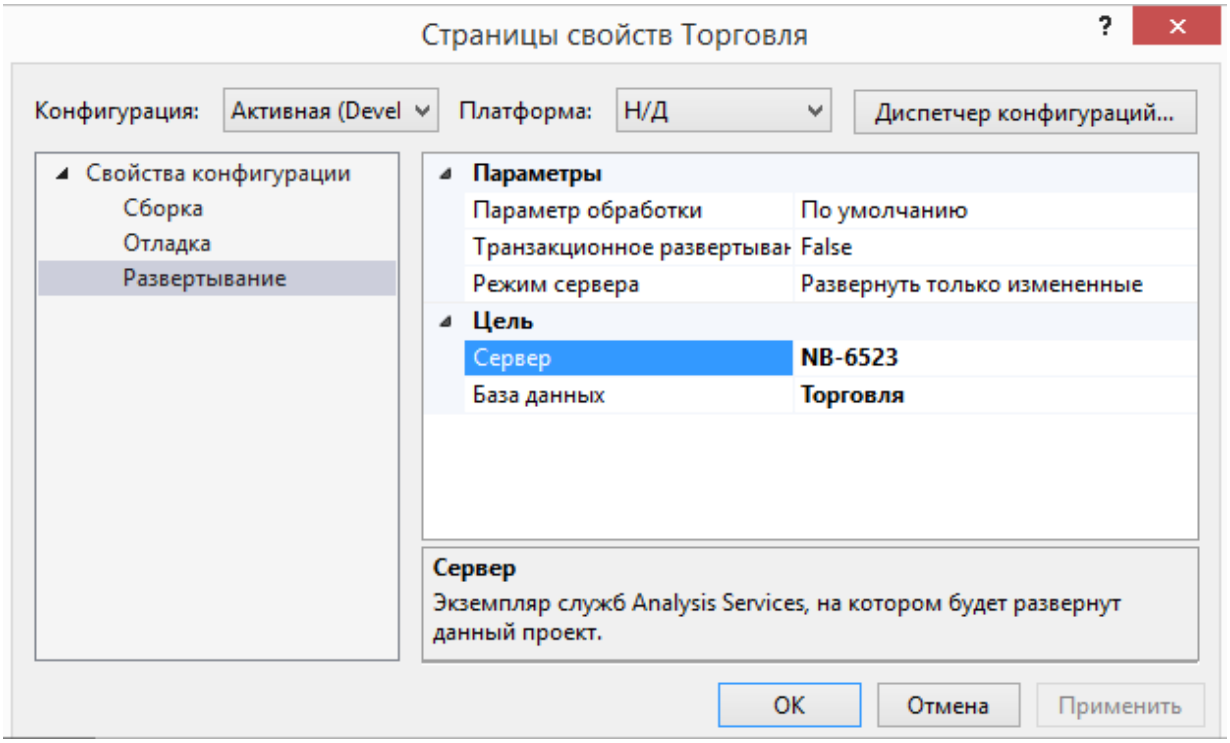


Рис. 31. Параметры развертывания проекта

7.5. Источники и представления исходных данных

Все начинается с описания источников исходных данных для создаваемых кубов. Для описания источника данных, размещенного на MS SQL сервере, нужно выбрать из контекстного меню элемента «Источники данных» команду «Создать источник данных» и с помощью мастера (Рис. 32) определить поставщика данных (на Рис. 32 это «Собственный поставщик OLE DB\SQL Server Native Client 11.0») и в случае сервера баз данных указать имя сервера (NB-6523) и базы (Торговля).

Далее необходимо задать представление источника данных (или несколько представлений). Команда «Создать представление источника данных…» контекстного меню запускает соответствующий мастер, позволяющий выбрать таблицы со значениями показателей и метками измерений. Представление (Рис. 33) имеет вид подсхемы реляционной базы данных с указанием ограничений ссылочной целостности. Для примера с Рис. 33 таблица «ТоварыВдоговорах» содержит описание продаж, по которому можно вычислить основные показатели «Количество», «Вес», «Стоимость» проданного товара. Другие таблицы можно использовать для построения измерений «Товары», «Продавцы», «Покупатели», «Договоры».

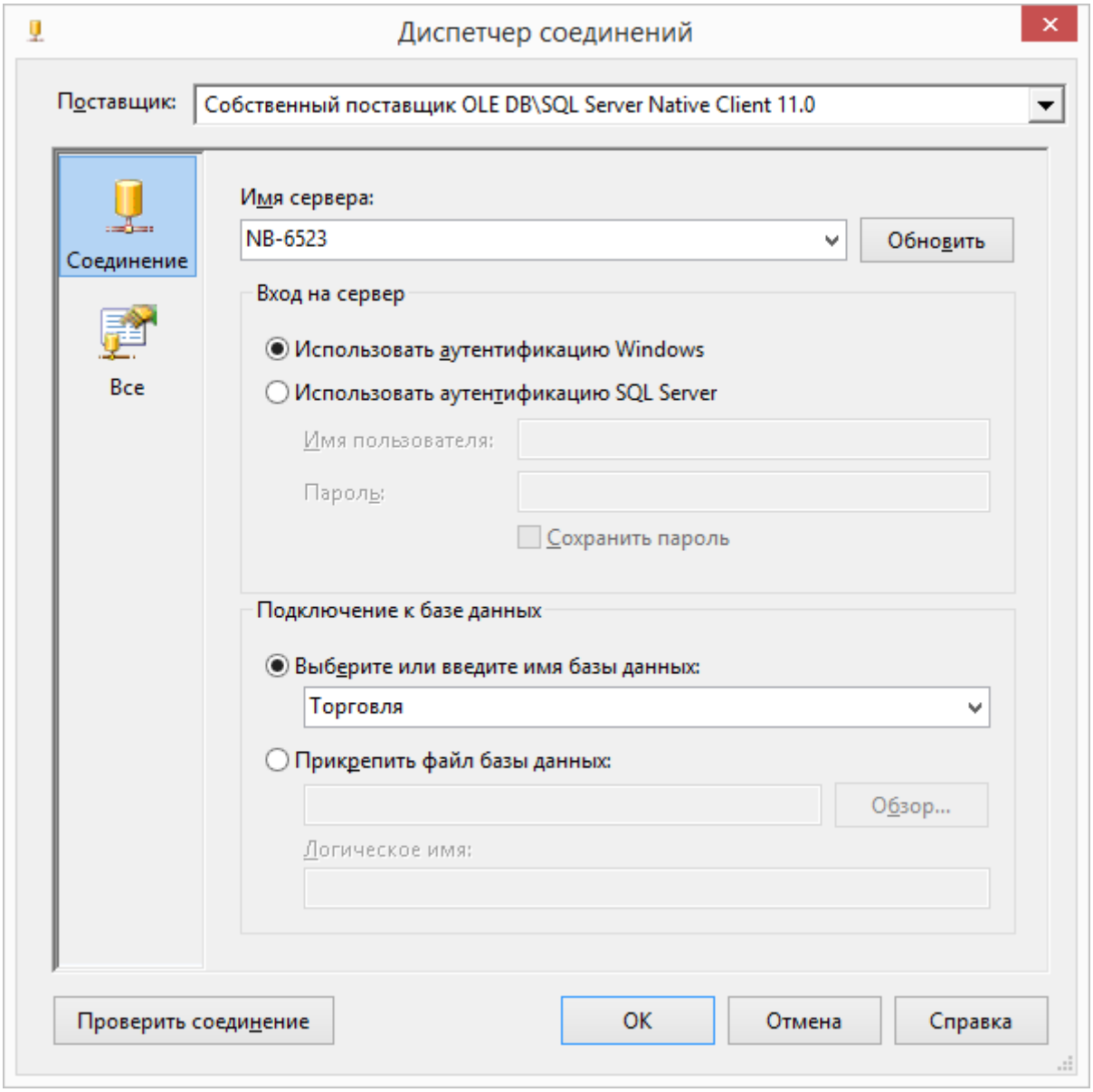


Рис. 32. Определение источника данных

Обычно таблицы реляционной БД не содержат всех показателей – их нужно вычислить. Для вычисления стоимости нужно создать (Рис. 34) в представлении таблицы «ТоварыВдоговорах» именованное вычисления соответствующей контекстной командой.

Аналогично создадим вычисляемое поле «Вес», определяемое выражением «Количество*(SELECT [Товары].[Вес ЕдИзм(Кг)] FROM [Товары] WHERE [dbo].[Товары].[Код товара] = [dbo].[ТоварыВдоговорах].[Код товара])».

В таблице Договоры определим вычисление полей «Год», «Месяц», «День» выражениями «Year(Дата)», «Month(Дата)», «Day(Дата)». Любую таблицу можно просмотреть командой «Просмотр данных» контекстного меню таблицы и проконтролировать правильность вычислений.

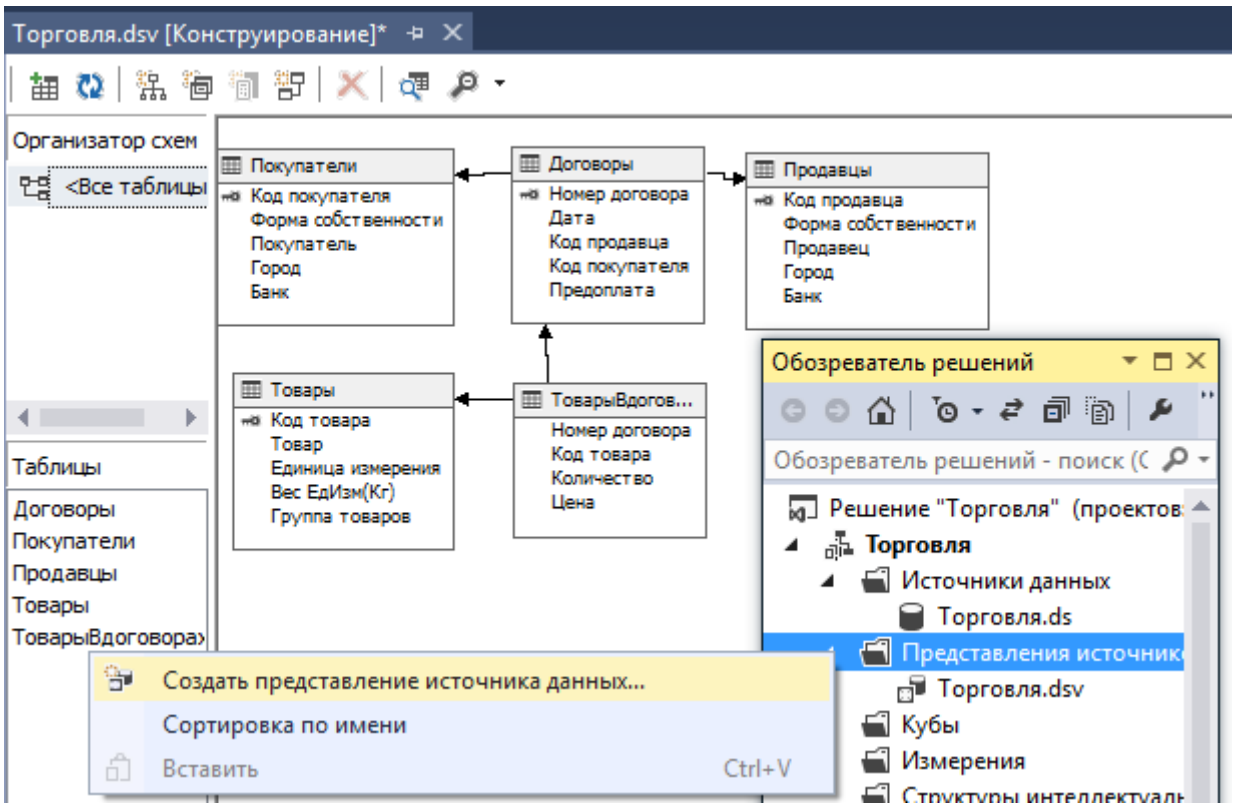


Рис. 33. Представление источника данных

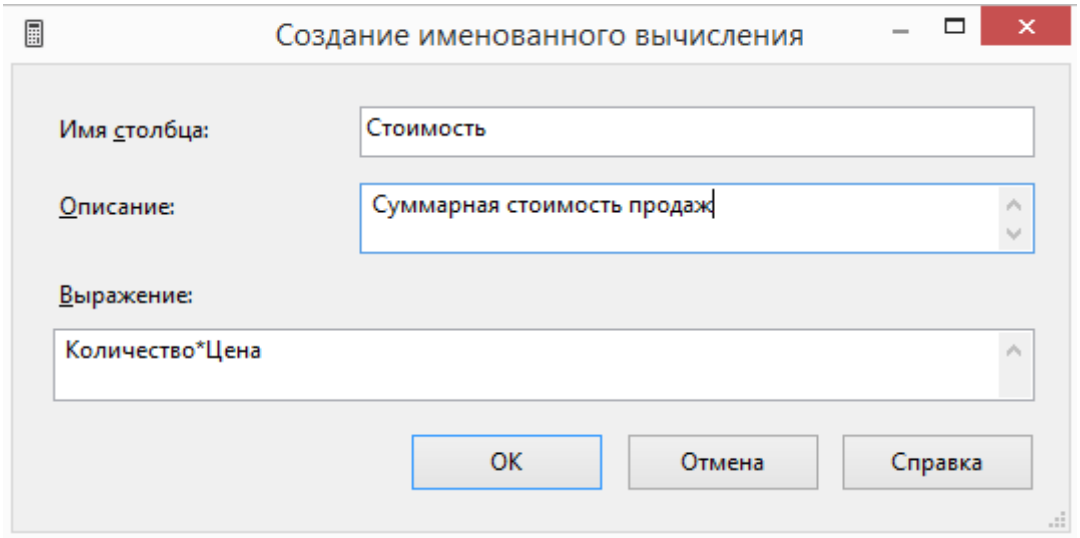


Рис. 34. Создание именованного вычисления

В представлении можно выделить таблицу фактов (Fact Table), которая содержит значения показателей (или данные для их вычисления) для комбинации меток измерений, обычно представленных кодами. В примере с Рис. 33 такой таблицей является «ТоварыВдоговорах». Данные для каждого измерения берутся из соответствующего справочника (Dimension Table), в котором значению кода соответствует метка измерения и другие атрибуты. Определение кубов начинается с создания измерений.

7.6. Создание измерений

Измерения создаются мастером, который запускается командой «Создать измерение» для вкладки «Измерения». В мастере выбирается метод создания – на основе существующей таблицы. Затем выбирается таблица, ключевые столбцы, столбец с именем метки (Рис. 35) и атрибуты метки (Рис. 36). В завершение работы мастера определяется имя измерения.

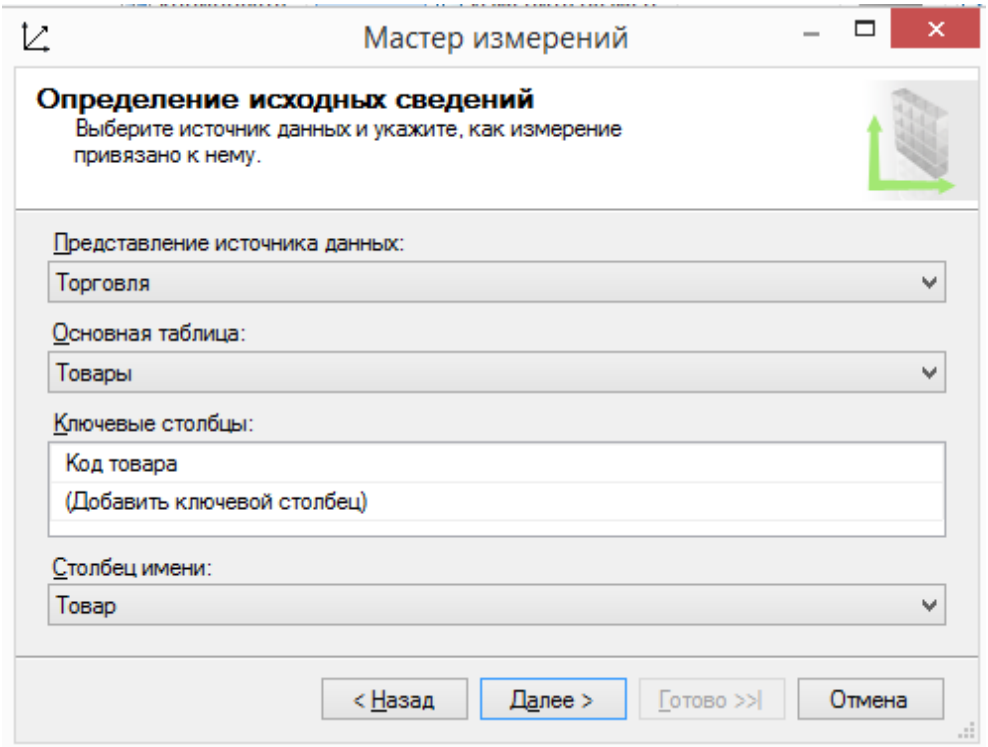


Рис. 35. Определение ключа и имени метки измерения

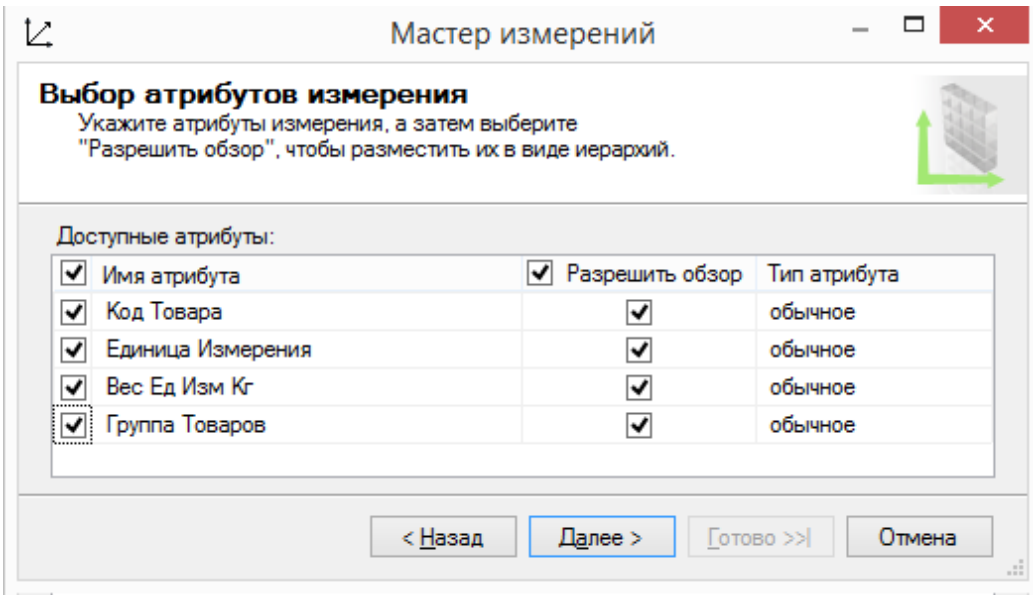


Рис. 36. Выбор атрибутов метки измерения

Для измерения можно определить одну или несколько иерархий. Например, для измерения «Товары» можно ввести двухуровневую иерархию «Группы товаров – товары». Это выполняется перетаскиванием полей в область определения иерархии (Рис. 37).

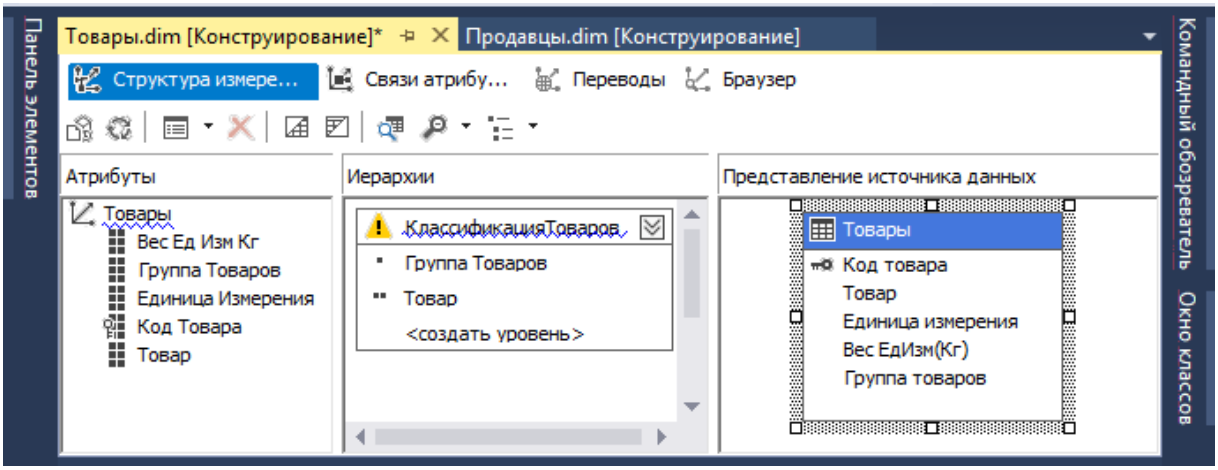


Рис. 37. Создание иерархии измерения

Возможно создание несбалансированных иерархий – типа «родитель – потомок» (parent – child). Такие иерархии нередко основаны на таблицах, где первичный ключ является одновременно и внешним ключом (например, для каждого работника указывается код его начальника).

Для появления измерения в многомерной БД необходимо выполнить развертывание (структуры из проекта переносятся в БД) и обработку (данные из источника копируются в БД) с помощью одноименных команд. После этого можно просматривать измерение (Рис. 38).

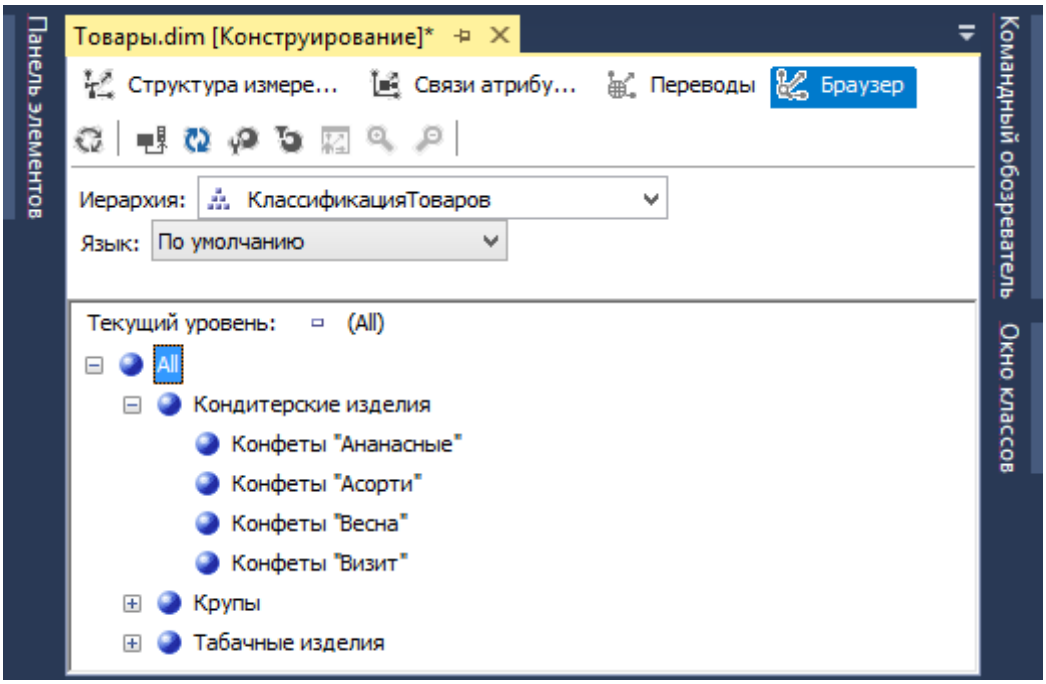


Рис. 38. Просмотр измерения

Аналогично создаются другие измерения: «Продавцы», «Покупатели», «Договоры».

7.7. Определение OLAP-кубов

Куб заполняется значениями показателей на основании таблицы фактов, содержащей показатели (меры) и связанной с таблицами измерений. Мастер создания куба (Cube wizard) включает выбор таблицы фактов (Рис. 39). Для рассматриваемого примера это таблица «ТоварыВдоговорах».

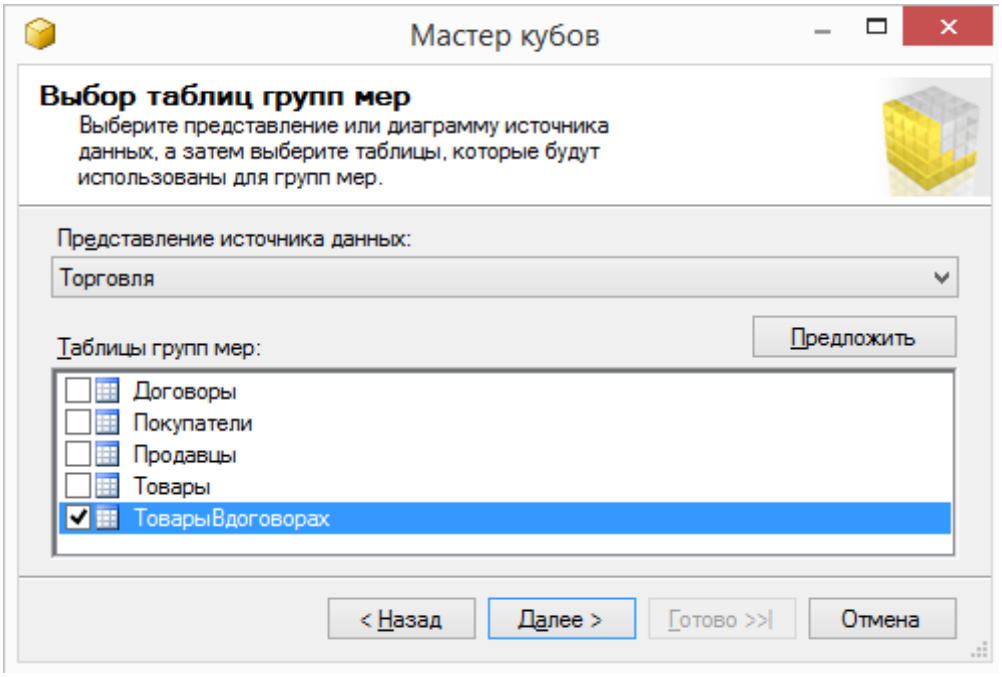


Рис. 39. Выбор таблицы фактов

Далее среди числовых полей таблицы фактов выбираются (Рис. 40) показатели (меры). Среди них система предлагает дополнительно число записей – «Число Товары Вдоговорах».

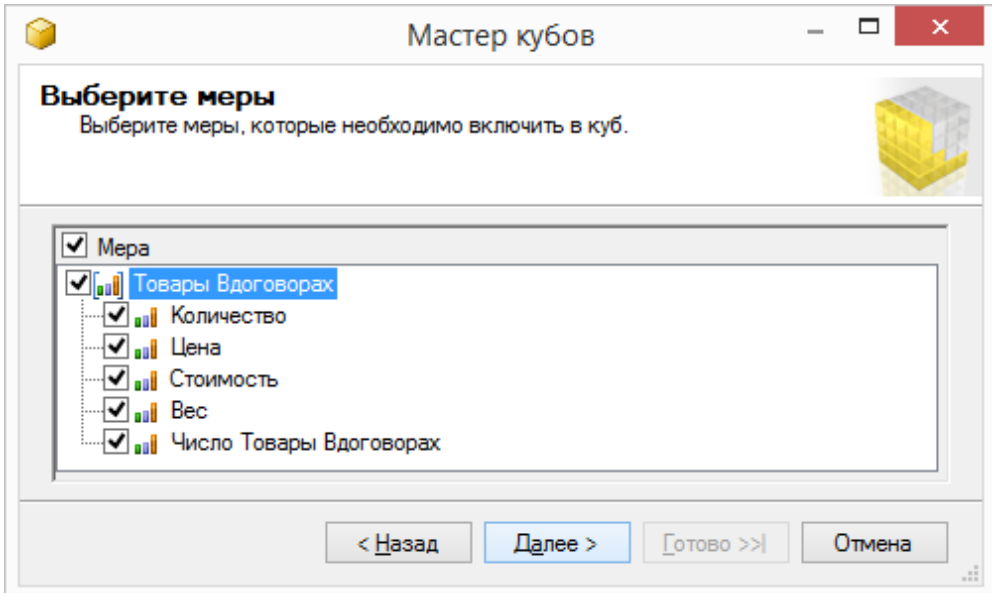


Рис. 40. Выбор показателей (мер)

После выбора показателей мастер предлагает выбрать измерения (Рис. 41).

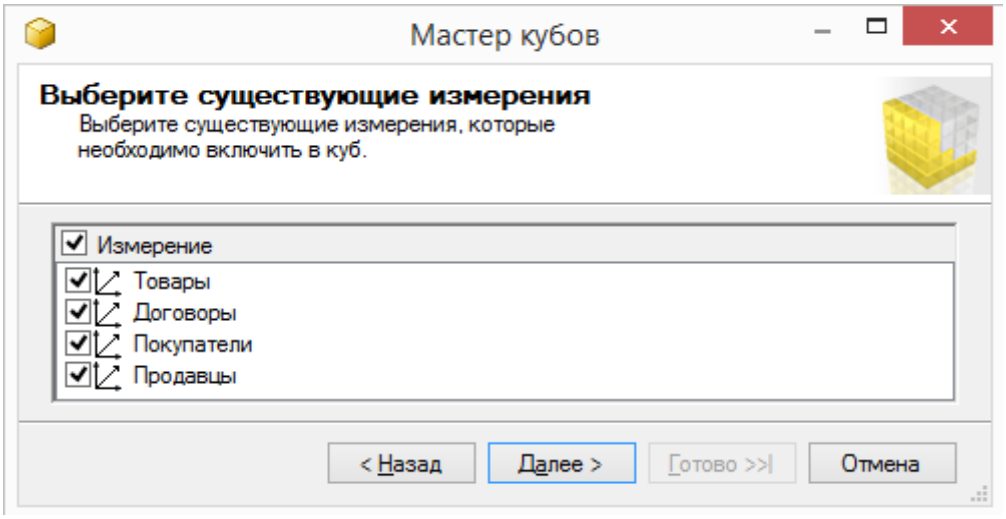


Рис. 41. Выбор измерений

Заканчивается работа мастера определением имени куба. В результате будет создано определение куба, на вкладке «Структура куба» будут представлены основные компоненты определения (Рис. 42). Для каждого показателя (кроме числа записей) будет задана функция агрегирования Sum – для формирования агрегированных значений показателя, вычисляемого по множеству ячеек. Это не всегда правильно. В рассматриваемом примере сумма цен не будет суммарной стоимостью, а следовательно, будет вводить пользователей в заблуждение. Для показателя «Цена» можно было бы задать усреднение в качестве функции агрегирования, однако и это будет некорректно. Правильно определить среднюю цену как суммарную стоимость, деленную на суммарное количество.

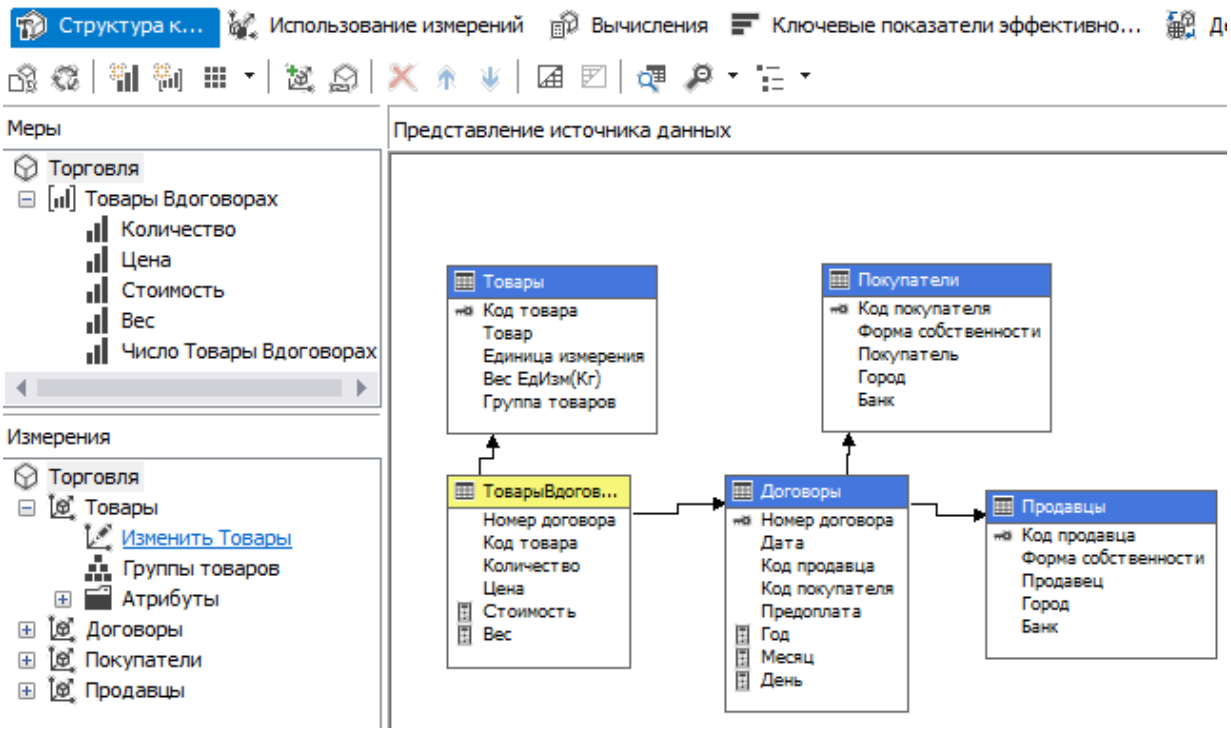


Рис. 42. Структура куба

Для задания другой функции агрегирования достаточно открыть свойства показателя. На Рис. 43 показано изменение в окне свойств функции агрегирования с Sum на Min и имени показателя «Цена» на «МинЦена».

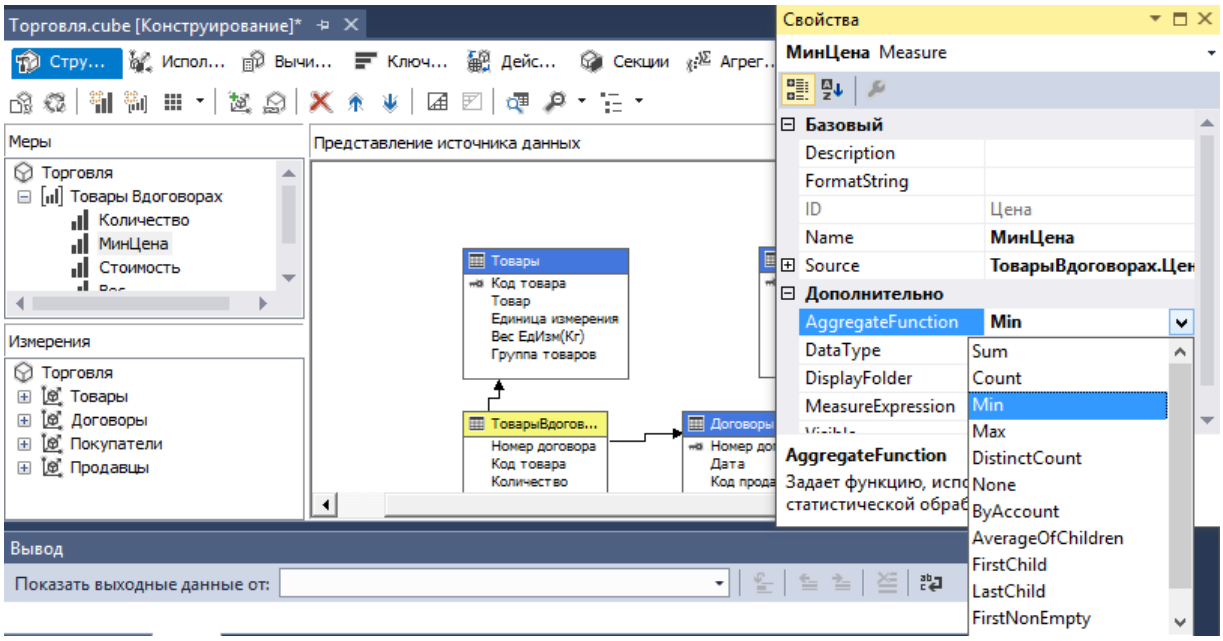


Рис. 43. Изменение свойств показателя

Для создания показателя, вычисляемого по другим показателям нужно перейти на вкладку «Вычисления» и выполнить команду контекстного меню «Создать вычисляемый элемент». На Рис. 44 показано определение средней цены в виде частного от деления суммарной стоимости на суммарное количество. Обратите внимание на идентификацию показателей в выражении. В процессе определения куба формируется специальное измерение – [Measures], которое содержит в виде меток имена показателей. Именно это именование и нужно использовать в формулах, это можно выполнить перетаскиванием имени показателя в поле «Выражение». Вычисляемые показатели не хранятся в кубе, а вычисляются по мере необходимости.

Мастер создания кубов создает простую структуру куба, в которой каждое измерение связано непосредственно с таблицей фактов. Такую схему называют «звезда». Измерения «Покупатели» и «Продавцы» связаны с таблицей фактов через измерение «Договоры». Такую схему называют «снежинка» и она не генерируется мастером создания куба. Такие связи нужно определять явно на вкладке «Использование измерений» (Рис. 45). Для рассматриваемого примера на этой вкладке будут только измерения «Товары» и «Договоры», связанные с таблицей фактов по схеме «Звезда».

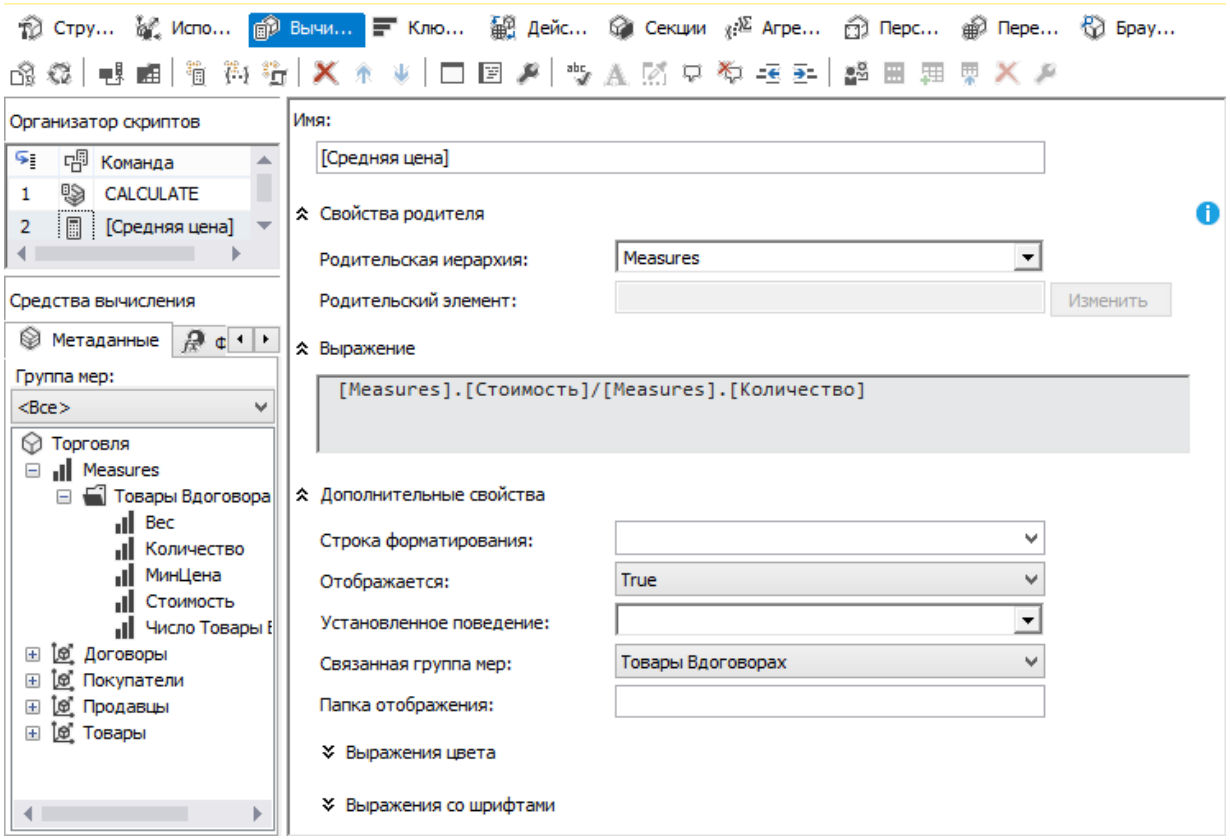


Рис. 44. Создание вычисляемого показателя

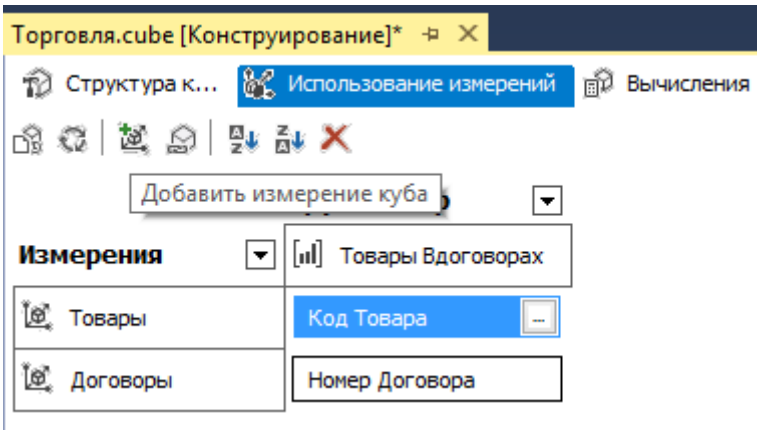


Рис. 45. Страница «Использование измерений»

Для добавления измерения можно воспользоваться одноименной кнопкой или командой контекстного меню. После выбора измерения нужно задать параметры связи (Рис. 46): тип связи – «ссылочный» (соответствующей схеме «снежинка»), промежуточное измерение – «Договоры», атрибуты связи – «Код Покупателя» и для выбранного измерения, и для промежуточного.

Аналогично производится добавление измерения «Продавцы». В результате на странице «Использование измерений» должны появиться все измерения куба (Рис. 47).

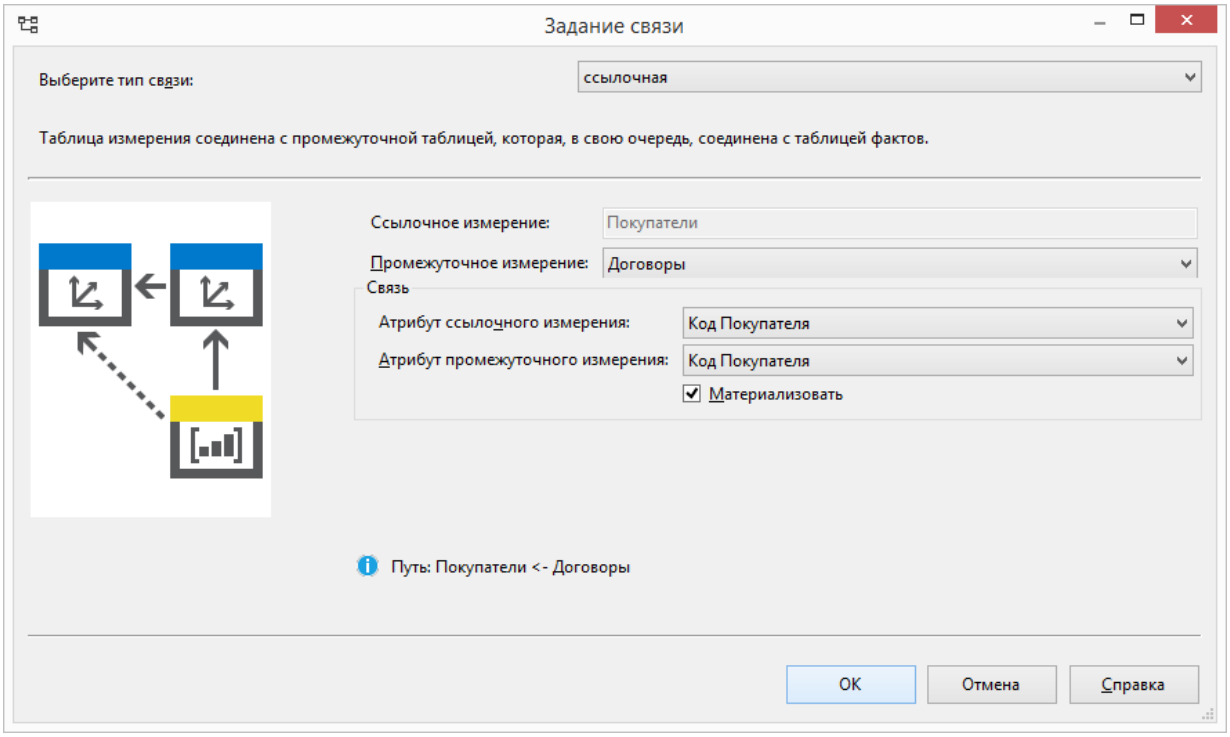


Рис. 46. Определение связи измерения и таблицы фактов

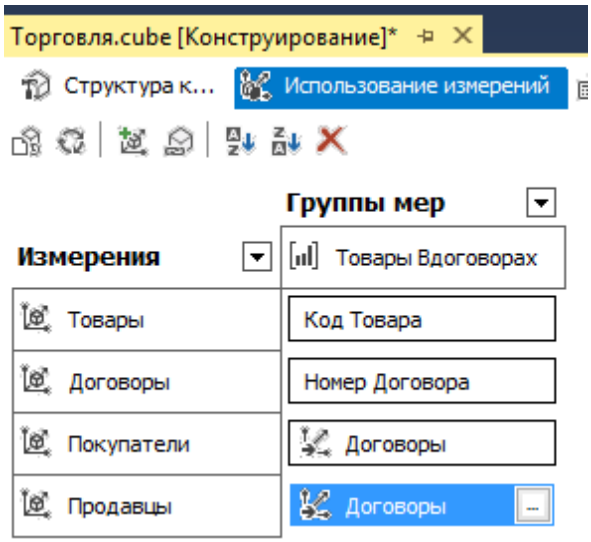


Рис. 47. Связи измерений и таблицы фактов

Чтобы использовать многомерную базу данных OLAP, сначала необходимо обработать ее кубы. Обработка куба – это его заполнение реальными данными из источника данных, организация вычислений и хранения агрегированных значений (агрегатов, обобщений).

Вычисления могут выполняться либо во время обработки куба, либо когда клиентское приложение запросит эти данные. Если создать агрегаты, то вычисления будут выполняться во время обработки куба и итоговые данные будут сохранены в базе данных OLAP.

Когда клиентское приложение запросит данные из куба, сначала будут просмотрены все сохраненные агрегаты, чтобы выяснить, существуют ли необходимые данные. Обнаружив запрошенные данные, сервер просто вернет их клиенту, не делая никаких дополнительных вычислений. Если же запрошенных данных нет ни в одном из сохраненных обобщений, то сервер должен вычислить данные «на лету». Используя агрегаты, можно существенно повысить производительность.

Однако агрегаты не лишены недостатков. Они занимают значительное место на сервере и могут потребовать много времени для обработки. Поэтому, создавая для куба обобщения, нужно найти компромисс между производительностью и памятью.

Для хранения используют секции. Секции представляют собой физическое место хранения исходных и агрегатов куба. При создании куба аналитические службы автоматически создают одну секцию. После создания куба можно вернуться к этому процессу и создать новые секции. Секции используются для физического сегментирования данных из куба.

Преимущество секций заключается в том, что у каждой из них может быть свой режим хранения и уникальный набор агрегатов. Таким образом, один логический куб можно разделить на несколько источников физических данных, выбрав для каждого из них режим хранения и набор агрегатов. Управление секциями куба выполняется в конструкторе куба на вкладке «Секции» (Рис. 48).

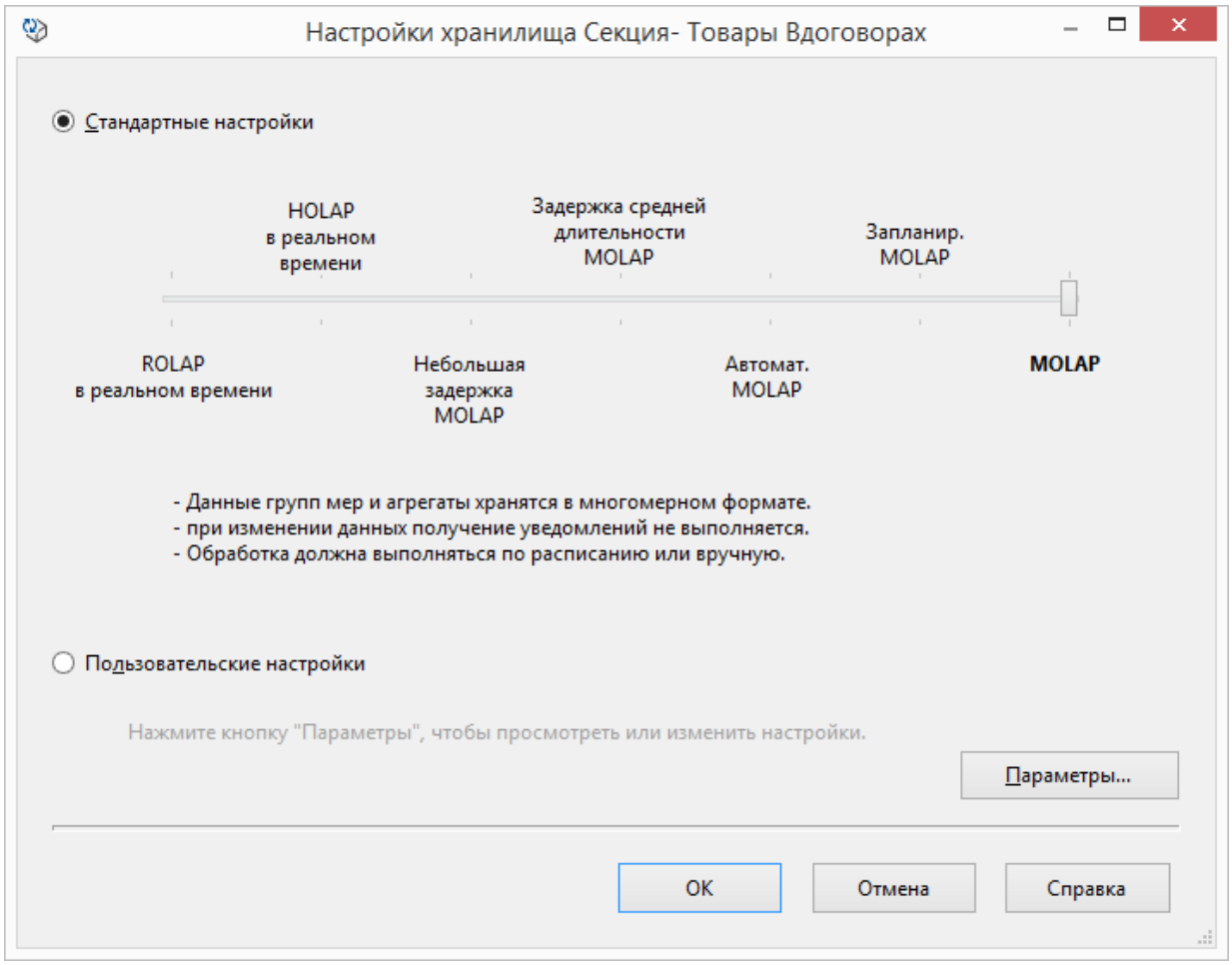


Рис. 48. Параметры хранения данных и агрегатов куба

Предусмотрены следующие режимы хранения:

1. ROLAP (Relational OLAP) реального времени. Исходные данные и агрегаты хранятся в реляционном формате. Сервер осуществляет прослушивание уведомлений при измерении данных, и все запросы отражают текущее состояние данных (нулевая задержка). Однако этот метод может давать самое большое время отклика.

2. HOLAP (Hibrid OLAP) реального времени. Исходные данные хранятся в реляционном формате, в то время как агрегаты хранятся в многомерном формате. Сервер осуществляет прослушивание уведомлений при измерении данных и обновляет агрегаты по необходимости. При обновлении источника данных сервер переключается на реляционный OLAP (ROLAP) реального времени до обновления агрегатов. Все запросы отражают текущее состояние данных (нулевая задержка). Этот метод обычно обеспечивает лучшую общую производительность по сравнению с хранилищем ROLAP.

3. MOLAP (Multidimensional OLAP) с малой задержкой. Исходные данные и агрегаты хранятся в многомерном формате. Сервер осуществляет прослушивание уведомлений об изменениях данных и переключается в режим ROLAP реального времени на время, пока объекты MOLAP повторно обрабатываются. Перед обновлением требуется интервал бездействия не менее 10 секунд. Если интервал бездействия не соблюдается, то активируется 10-минутный интервал прерывания. Обработка осуществляется автоматически при изменениях данных с целевой задержкой, равной 30 минутам после первого изменения. Эта настройка обычно будет использоваться для источника данных с частыми обновлениями, для которого производительность запросов является более важной, чем постоянное предоставление самых последних данных. Эта настройка автоматически обрабатывает объекты MOLAP при необходимости, после интервала задержки. Во время повторной обработки объектов MOLAP снижается производительность.

4. MOLAP со средней задержкой. Технология хранения соответствует MOLAP с малой задержкой, за исключением задержки, равной четырем часам.

5. Автоматический MOLAP. Исходные данные и агрегаты хранятся в многомерном формате. Сервер осуществляет прослушивание уведомлений, но сохраняет самый последний кэш MOLAP при построении нового. Сервер никогда не переключается в режим OLAP реального времени, и запросы могут выдавать устаревшие данные при построении нового кэша. Обработка осуществляется автоматически при измерениях данных с целевой задержкой, равной двум часам. Эта настройка обычно используется для источника данных, для которого производительность запросов является ключевым фактором. Запросы не возвращают самые последние данные во время построения и обработки кэша.

6. Запланированный MOLAP. Исходные данные и агрегаты хранятся в многомерном формате. Сервер не получает уведомлений об изменении данных. Обработка осуществляется автоматически каждые 24 часа. Эта настройка

обычно используется для источника данных, в котором необходимы только ежедневные обновления. Запросы всегда осуществляются в отношении данных в кэше MOLAP, который не очищается до тех пор, пока не будет построен новый кэш и его объекты не будут обработаны.

7. MOLAP. Упреждающее кэширование не включено. Исходные данные и агрегаты хранятся в многомерном формате. Сервер не получает уведомлений об изменении данных. Обработка должна либо быть запланирована, либо осуществляться вручную. Эта настройка обычно используется для источника данных, в котором периодические обновления не важны для клиентских приложений, но для которого высокая производительность является критической.

Режим хранения выбирается, исходя из требований к оперативности и производительности многомерного анализа. Высокая производительность достигается за счет частичной потери оперативности.

7.8. Обеспечение безопасности данных OLAP

Аналитические службы предоставляют пользователям доступ к данным, предусматривающий проверку полномочий. Полномочия определяются для ролей (role), членами которых могут быть учетные записи (login) пользователей компьютерной системы. Пользователь получает полномочия тех ролей, в которые он включен.

После установки аналитических служб создается локальная группа OLAP Administrators. По умолчанию учетная запись пользователя, от имени которой устанавливалась аналитические службы, становится членом этой группы. Все пользователи этой группы могут выполнять административные функции, в частности создавать новые роли, включать в них пользователей и предоставлять полномочия.

Роли определяются на уровне базы данных и отображаются в обозревателе. Добавить роль и пользователей можно с помощью контекстного меню. На Рис. 49 для базы данных «Торговля» определена роль «Отдел сбыта» и на вкладке «Членство» в нее включены пользователи с логинами user и seleznev.

Основные полномочия определяются для доступа к данным куба. Для каждой роли на вкладке «Кубы» (Рис. 50) можно определить для каких кубов пользователи роли могут выполнять чтение данных.

Указание куба означает доступ ко всем данным куба. Эти полномочия можно ограничить отдельными измерениями на вкладке «Измерения» (Рис. 51).

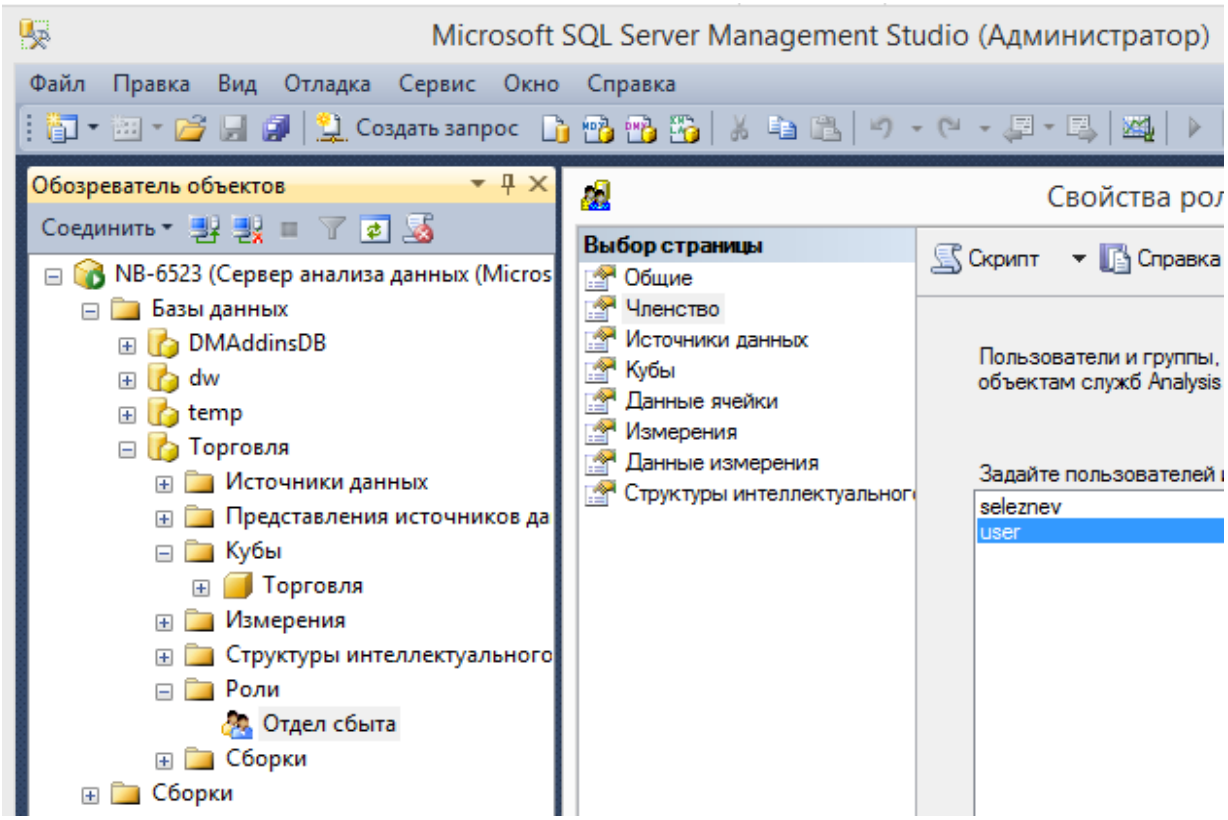


Рис. 49. Роль и ее состав

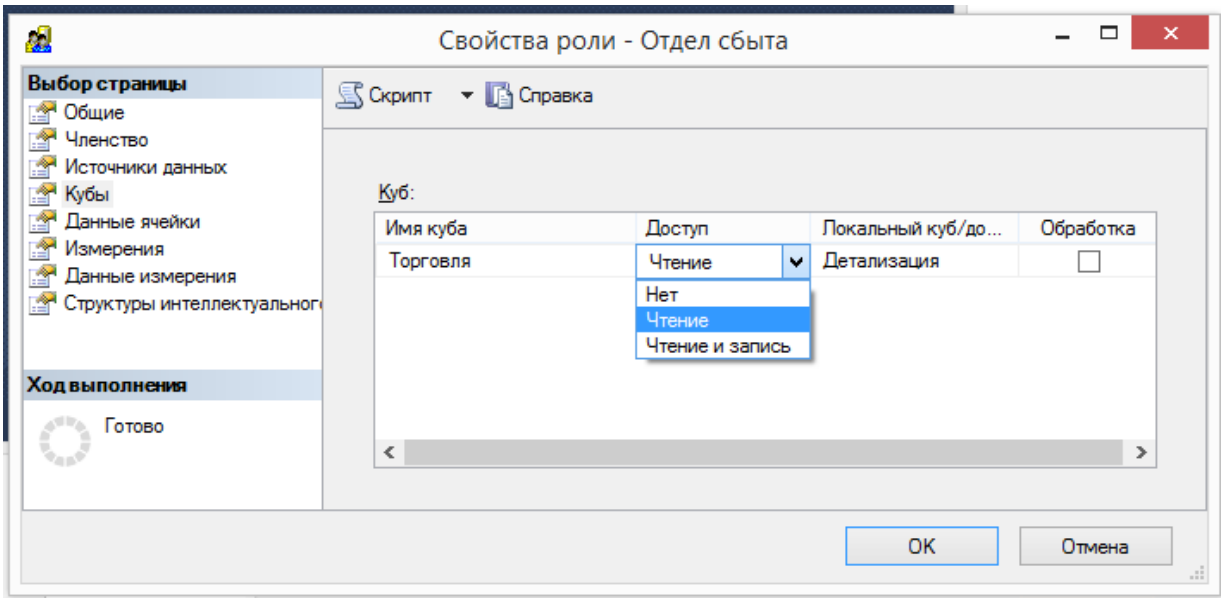


Рис. 50. Определение доступа членов роли «Отдел сбыта» к кубу

Еще одна возможность – ограничение доступа данными, которые соответствуют отдельным меткам некоторого измерения – предоставляется на вкладке «Данные измерения». На Рис. 52 продемонстрировано как можно выбрать для доступа определенные показатели (меры). Все показатели включаются в специфическое измерение «Measures» мер, создаваемое автоматически для каждого куба.

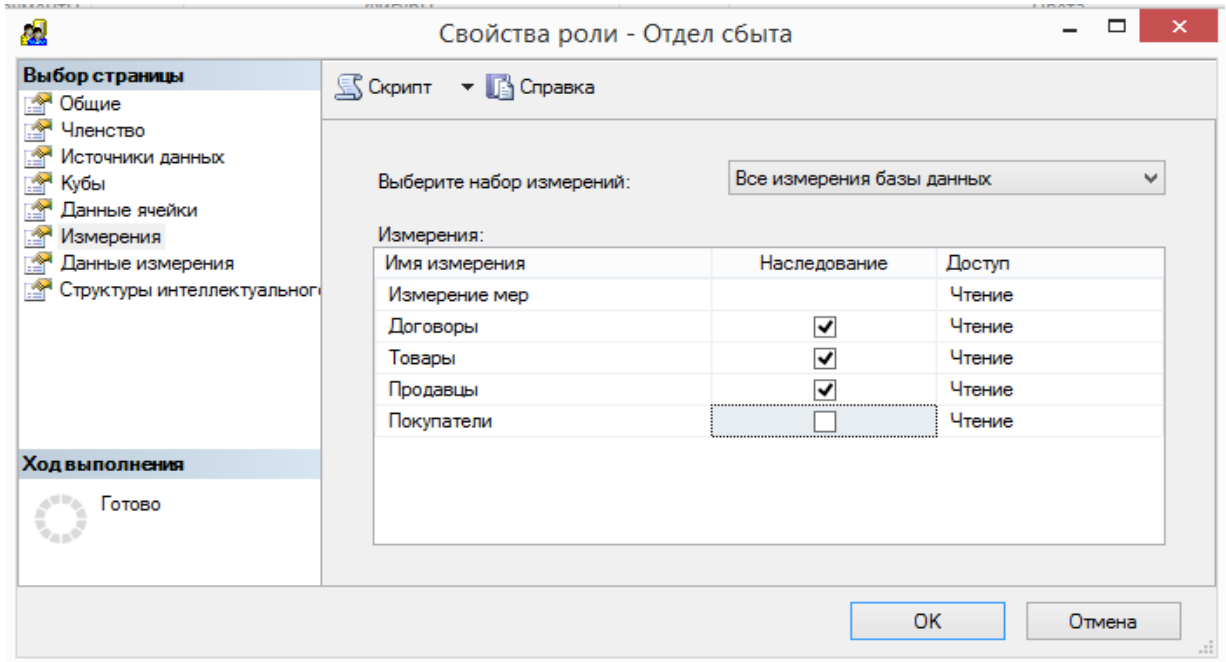


Рис. 51. Указание доступа к отдельным измерениям

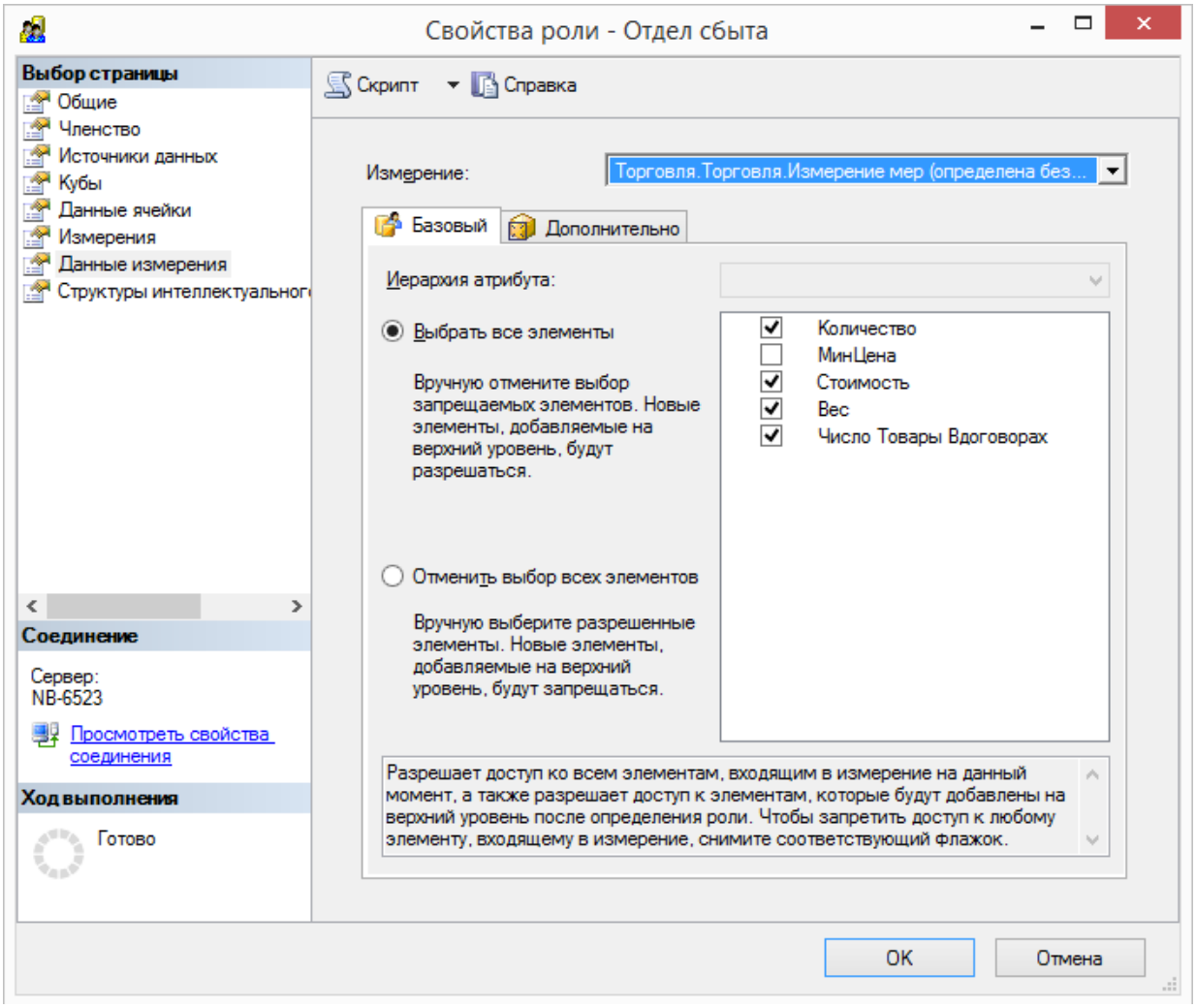


Рис. 52. Ограничение данных метками измерения

7.9. Клиенты OLAP-данных

Систематический сбор и накопление данных позволяют собрать огромный по объему материал для аналитической обработки. Доступ к этим данным может быть обеспечен на уровне специализированных программ, или в среде офисных приложений, или через Интернет. Наиболее трудоемкий процесс – разработка специализированных программ доступа к хранилищам данных, и одновременно программы могут обеспечить наибольший уровень сервиса для пользователей. Тем не менее и другие варианты доступа являются достаточно удобными.

Доступ к OLAP-данным в Microsoft Excel

MS Excel сам по себе обладает OLAP-функциональностью в виде сводных таблиц, являющихся аналогом кубов. Поэтому именно эта часть MS Office обеспечивает не только доступ к данным куба, но может применяться для анализа данных с помощью операций среза, агрегирования и детализации. Все эти операции выполняются в привычном интерфейсе для пользователей, умеющих работать со сводными таблицами. Они могут работать с серверными кубами как с локальными сводными таблицами, даже не подозревая, что все вычисления выполняет не MS Excel, а аналитические службы на сервере.

Для построения сводной таблицы по данным хранилища в Microsoft Excel можно воспользоваться мастером сводных таблиц (команда «Данные», «Сводная таблица»):

1. Для доступа к данным из хранилища следует выбрать вариант «во внешнем источнике данных».

2. Для получения данных следует нажать кнопку «Выбрать подключение». Если нужного подключения нет в списке, то следует выбрать кнопку «Найти другие» и создать новое подключение соответствующим мастером:

   2.1. Выбрать тип источника «Microsoft SQL Server Analysis Services».

   2.2. Ввести имя сервера и определить способ аутентификации.

   2.3. Выбрать базу данных и куб.

   2.4. Определить имя подключения и сохранить файл подключения.

3. Далее выполняются обычные действия по определению макета сводной таблицы.

В результате пользователь получает готовый Excel-файл (Рис. 53), обеспечивающий работу с кубом, расположенным на сервере и регулярно обновляемым.

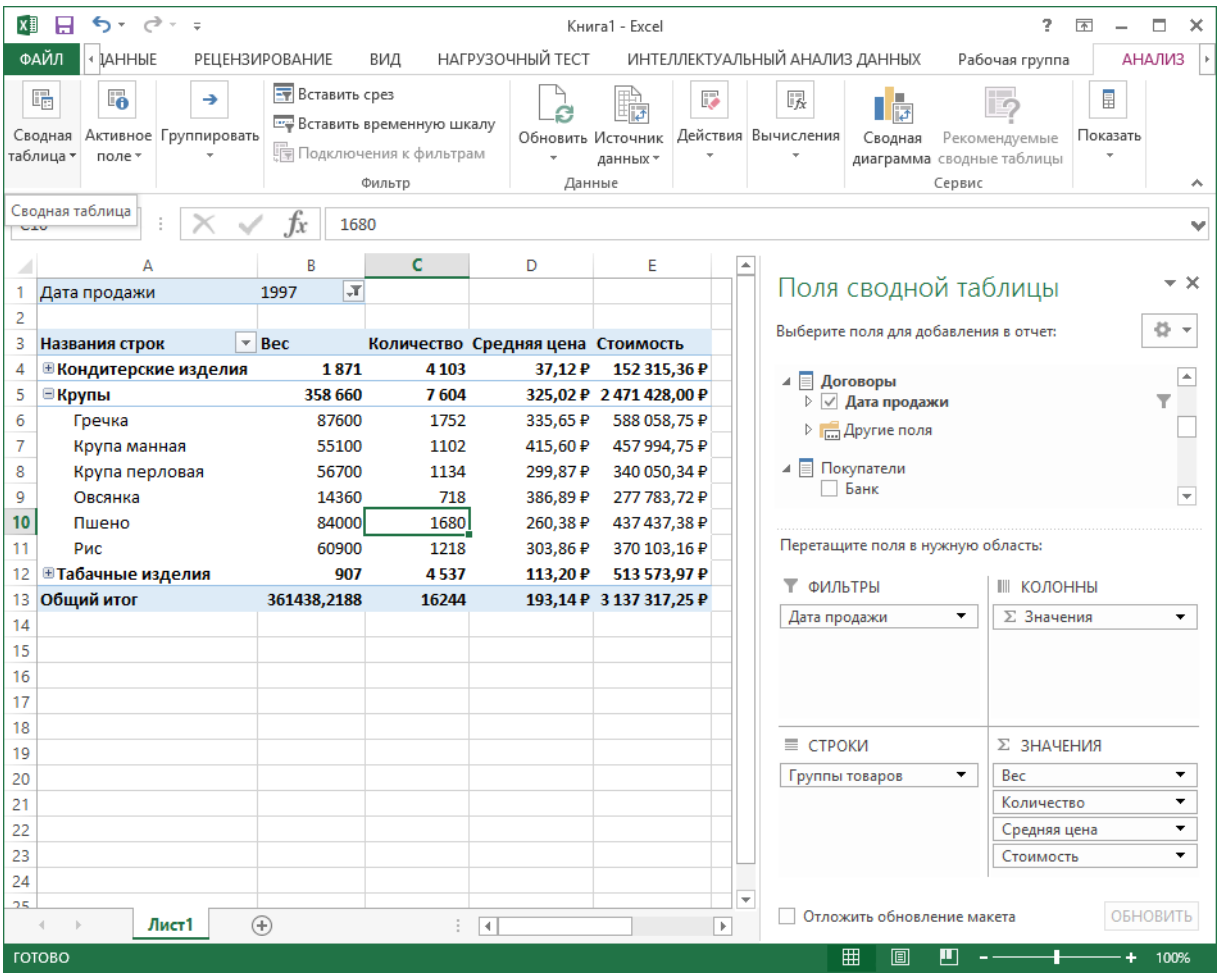


Рис. 53. Доступ к кубу на сервере с помощью MS Excel

Публикация сводных таблиц на веб-страницах

Самый простой способ построить веб-страницу с OLAP-функциональностью заключается в использовании компонента PivotTable List. Для этого нужно сохранить сводную таблицу Microsoft Excel с доступом к кубу (см. выше) как веб-страницу. Для этого выберем в Microsoft Excel пункт меню «Файл», «Сохранить как веб-страницу», в появившейся диалоговой панели выберем переключатель «Добавить интерактивность», нажмем кнопку «Опубликовать», в диалоговой панели выберем из выпадающего списка «Элементы «Лист1» и добавим «Работу со сводными таблицами».

Далее следует изменить заголовок, который появится на будущей Web-странице, и сохранить ее. Если открыть эту страницу в Microsoft Internet Explorer версии 4.01 или выше, мы увидим, что она содержит PivotTable List – элемент управления, предназначенный для просмотра OLAP-данных и сводных таблиц на веб-страницах.

Сразу же заметим, что этот элемент управления можно применять только в локальных сетях на компьютерах, для которых приобретена лицензия на Microsoft Office; другие способы его применения, например на веб-страницах, доступных в Интернете, запрещены лицензионным соглашением.

Пользователь, манипулирующий PivotTable List в браузере или в Windows-приложении, может, как и в сводной таблице Excel, перемещать данные в область строк, столбцов и страниц (в Microsoft Office Web Components приняты термины Row Area, Column Area и Filter Area) с диалоговой панели, напоминающей панель «Список полей сводной таблицы» из Excel.

Пользователь может также выполнять операцию детализации (drill-down), щелкая мышью на значках «+». Компонент PivotTable List позволяет сортировать и фильтровать данные. Во-первых, фильтрация данных может быть осуществлена с помощью отображения только выбранных членов измерений, которые могут быть отмечены в выпадающем списке, сходном с соответствующим списком Excel. Во-вторых, с помощью диалоговой панели «Команды и параметры» можно выбрать способы фильтрации и группировки данных.

Помимо этого, пользователь может изменять атрибуты отображения данных: цвет и шрифт текста, цвет фона, выравнивание текста, отображение и т.д. Для этого достаточно поместить курсор на один из элементов данных, атрибуты которых нужно изменить (например, на наименование члена измерения, на ячейку с суммарными данными или с итоговыми значениями), и выбрать новые атрибуты отображения данных этого типа в той же диалоговой панели «Команды и параметры».

Кроме того, компонент PivotTable List позволяет на основе агрегированных данных вычислять доли или проценты общей суммы или суммы, соответствующей родительскому члену измерения (например, процент от годовой прибыли, полученный в данном квартале), – соответствующие опции можно найти в контекстных меню элементов данных.

Пользователю также доступен специально предназначенный для него файл справки (на русском языке, если используются веб-компоненты из комплекта поставки русской версии Microsoft Office XP). Однако пользователь не может изменить источник данных и отобразить на веб-странице другой OLAP-куб, поскольку право сделать это есть только у разработчика веб-страницы.

Отметим, что подобную веб-страницу можно создать и с помощью Microsoft FrontPage.

7.10. Язык запросов к многомерным данным MDX

Язык запросов к многомерным данным MDX (MultiDimensionaleXpressions) был впервые введен в рамках спецификации OLE DB for OLAP для работы с многомерными кубами. Будучи открытым стандартом, MDX является основным инструментом программирования для аналитических служб Microsoft SQL Server.

Для выполнения запросов на языке MDX можно использовать окно MDX-запросов в MS SQL Server Management Studio. Для создания окна можно указать многомерную базу данных и воспользоваться контекстным меню. На Рис. 54 для многомерной БД «Торговля» с помощью контекстного меню создается запрос для ввода и обработки многомерного выражения на MDX.

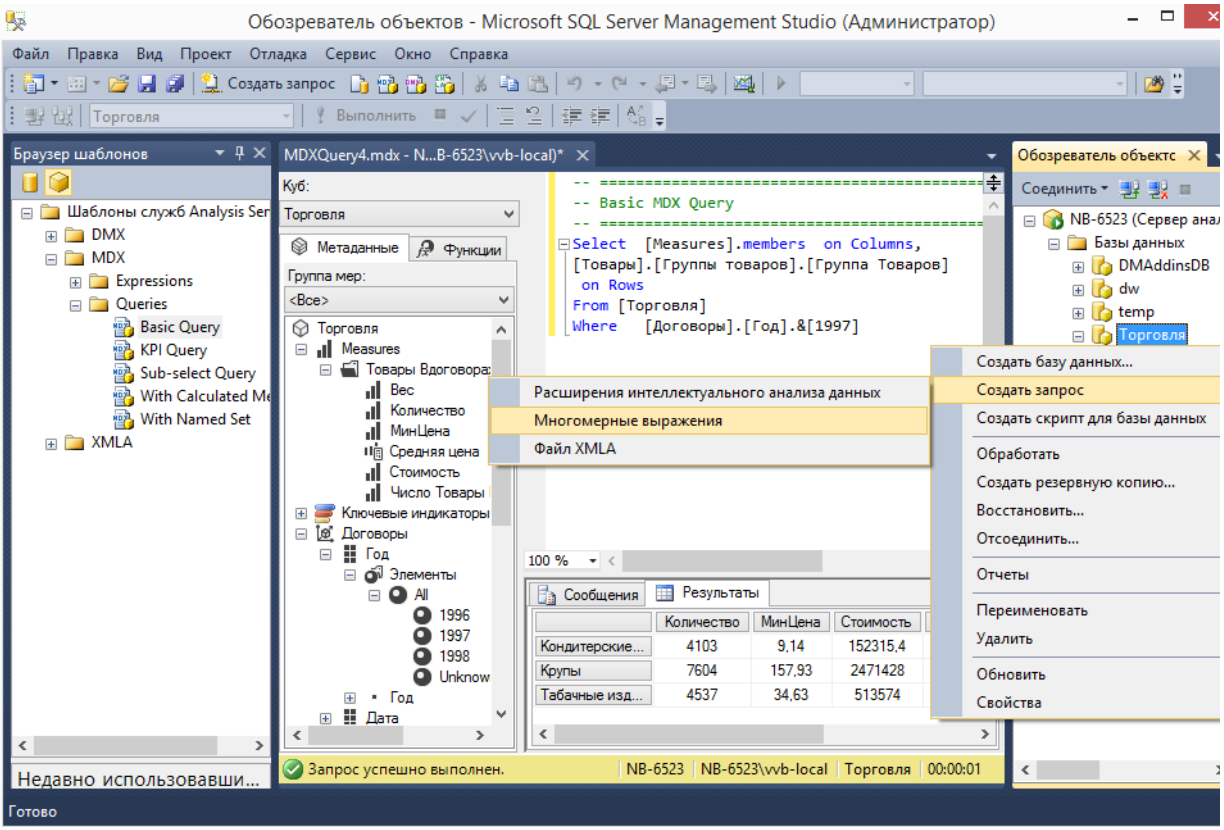


Рис. 54. Окно MDX-запросов в MS SQL Server Management Studio

Можно вводить MDX-команды непосредственно в панели запросов (3-я панель справа на Рис. 54) или конструировать запрос, перетаскивая измерения и меры куба в панель запросов из области (2-я панель) метаданных куба. Помимо этого, можно использовать примеры функций из панели «Браузер шаблонов». Выполнить запрос можно нажав кнопку «Выполнить» на панели инструментов. Результат выполнения запроса отображается в нижней части экрана.

MDX использует следующие понятия:

1. ***Компонент*** или ***метка*** (Member) – значение измерения на любом уровне иерархии. Определяется перечислением всех узлов, расположенных на пути от выбранной метки к вершине. Примеры:

[Товары].[КлассификацияТоваров]

[Товары].[КлассификацияТоваров].[Группа Товаров].&[Крупы]

[Товары].[КлассификацияТоваров].[Группа Товаров].&[Крупы].&[Рис]].

Для исключения двусмысленности имена заключают в квадратные скобки, если они содержат специальные знаки, такие как пробел. Символ амперсанда (&) перед именем метки указывает на ссылку по ключу. На каждую метку измерения можно ссылаться как по имени, так и по ключу, хотя рекомендуется все же последний вариант.

2. ***Кортеж*** (Tuple) – коллекция ячеек куба (срез), соответствующих некоторой комбинации значений измерений. Определяется как набор меток в круглых скобках. Размерность кортежа – множество измерений меток, вошедших в кортеж. Например, кортеж

([Measures].[Стоимость],
[Товары].[КлассификацияТоваров].[Группа Товаров].&[Крупы].&[Рис],
[Договоры].[Год].&[1997])

соответствует измерениям «Measures», «Товары», «Договоры».

3. ***Множество (Set)*** – набор кортежей. Определяется как перечисление значений (кортежей) в фигурных скобках. Размерности всех кортежей множества должны совпадать. Например, {[Продавцы].[Продавец].&[Вест], [Продавцы].[Продавец].&[Фокус]}.

Запрос на языке MDX представляет собой команду, которая выглядит следующим образом:

SELECT [<ось> [, <ось>...]]
FROM [<куб>]
[WHERE [<срез>]],

где *<ось>* – описание измерения куба-результата в виде *<множество> ON <имя оси>*, *<имя оси>* может быть следующим:

- *COLUMNS* – колонки;
- *ROWS* – строки;
- *PAGES* – страницы;
- *SECTIONS* – разделы;
- *CHAPTERS* – главы;
- *AXIS(<номер>)* – ось с указанным номером (оси нумеруются с нуля).

Срез куба задается кортежем. Одно измерение не может употребляться одновременно для задания двух осей или оси и среза. Для указания множества меток иерархии (или ее уровня) применяется метод members. Кортеж заключается в круглые скобки, в то время как множество – в фигурные. Порядок перечисления измерений и мер в кортеже не имеет значения.

Результатом запроса всегда является куб – значения одного или нескольких показателей, соответствующих значениям измерений (Рис. 55). Измерения располагаются по одной или нескольким осям, определенным в запросе. Например, выборка с Рис. 55 создается следующим запросом:

```
Select[Договоры].[Дата продажи].[Год] on Columns,
       [Продавцы].[Продавец].[Продавец] on rows
From [Торговля]
Where       [Measures].[Стоимость]
```

Сообщения | Результаты

	1996	1997	1998	Unknown
Вест	490371,3	273834,5	145867,5	(null)
Геракл	616712,4	358835,3	142352,9	(null)
Гермес	537810,8	267097,7	(null)	(null)
Единство	149437,1	197379	(null)	(null)
Норд	95125,13	483208	43964,01	(null)
Плюс	164509,2	105589,4	(null)	(null)
Ромашка	240157,6	267136,8	9080,44	(null)
СибАтом	106461,7	376934,3	12278,67	(null)
Тор	294353,5	256856,4	73893,12	(null)
Успех	263373,8	369430,3	5736,27	(null)
Федоров	(null)	(null)	(null)	(null)
Фокус	226937	181015,8	47303,55	(null)
Unknown	(null)	(null)	(null)	(null)

Рис. 55. Результат запроса MDX

Набором меток каждой оси можно управлять. Прежде всего можно образовывать метки перечислением. Например,

```
Select    {[Договоры].[Дата продажи].[Год].&[1996],
          [Договоры].[Дата продажи].[Год].&[1998]} on Columns,
{[Продавцы].[Продавец].&[Норд],
          [Продавцы].[Продавец].&[Успех]} on rows
From [Торговля]
Where          [Measures].[Стоимость]
```

В результате в каждом измерении будет две метки (Рис. 56).

	1996	1998
Норд	95125,13	43964,01
Успех	263373,8	5736,27

Рис. 56. Результат запроса с перечислением меток

Кроме этого, множество может быть задано интервалом (Рис. 57):
<1-ая метка> : <последняя метка>, например

```
Select    {[Договоры].[Дата                    продажи].[Год].&[1996]:
          [Договоры].[Дата продажи].[Год].&[1998]} on Columns,
{[Продавцы].[Продавец].&[Норд]:
          [Продавцы].[Продавец].&[Успех]} on rows
From      [Торговля]
Where     [Measures].[Стоимость]
```

	1996	1997	1998
Норд	95125,13	483208	43964,01
Плюс	164509,2	105589,4	(null)
Ромашка	240157,6	267136,8	9080,44
СибАтом	106461,7	376934,3	12278,67
Тор	294353,5	256856,4	73893,12
Успех	263373,8	369430,3	5736,27

Рис. 57. Результат запроса с заданием множества меток интервалом

Для определения осей можно использовать функцию *Crossjoin(<Множество>, <Множество>)*, которая позволяет комбинировать несколько измерений в одно измерение куба-результата. Запрос

```
Select {[Measures].[Количество], [Measures].[Стоимость],
[Measures].[Средняя цена]} on Columns,
CrossJoin({[Договоры].[Дата продажи].[Год].&[1996],
[Договоры].[Дата продажи].[Год].&[1998]},
  {[Продавцы].[Продавец].&[СибАтом]:
[Продавцы].[Продавец].&[Успех]}) on rows
From [Торговля]
```

формирует двухуровневую структуру (год продажи, продавцы) заголовков столбцов (Рис. 58).

		Количество	Стоимость	Средняя цена
1996	СибАтом	571	106461,7	186,447843695271
1996	Top	1567	294353,5	187,845265634971
1996	Успех	1469	263373,8	179,287780803268
1998	СибАтом	99	12278,67	124,026968907828
1998	Top	300	73893,12	246,310390625
1998	Успех	48	5736,27	119,505625406901

Рис. 58. Результат запроса с функцией CrossJoin

Filter((<Множество>, <Условия>) позволяет выполнять фильтрацию меток измерения (оси) – выбираются метки, удовлетворяющие условию.

Запрос

```
Select {[Measures].[Количество], [Measures].[Стоимость],
[Measures].[Средняя цена]} on Columns,
Filter([Продавцы].[Продавец].members, [Measures].[Стоимость]>500000) on
rows
From [Торговля]
```

выберет продавцов, у которых стоимость продаж больше 500000.

Order(<Множество>, <Выражение> [, ASC | DESC | BASC | BDESC]) сортирует метки измерения с сохранением иерархии *ASC | DESC* или без нее *BASC | BDESC*.

Ниже приведен запрос с сортировкой товаров по убыванию стоимости

```
Select {[Measures].[Количество],[Measures].[Стоимость]} on Columns,
  ORDER([Товары].[КлассификацияТоваров].members,
[Measures].[Стоимость], basc) on rows
From [Торговля]
```

В результате (Рис. 59) получится список всех меток измерения «Товары» отсортированных независимо от уровня иерархии метки. Применение ключа «*ASC*» приведет к сортировке сначала групп, а внутри каждой группы товаров.

	Количество	Стоимость
Конфеты "Весна"	2927	54437,46
Конфеты "Ананасные"	1840	71624,13
Конфеты "Асорти"	2489	77432,13
Конфеты "Визит"	2548	151652,8
Сигареты Петр 1	2360	174827,8
Сигареты LM	2485	300945,2
Сигареты Fine Line	2453	322485,7
Кондитерские изделия	9804	355146,5
Сигареты MarlBoro	2622	398807,4
Крупа перловая	2623	707153,3
Крупа манная	2040	848554,9
Овсянка	2337	887262,8
Пшено	3295	905662,6
Рис	2736	931824,3
Гречка	2801	970373,1
Табачные изделия	9920	1197066
Крупы	15832	5250831
All	35556	6803042

Рис. 59. Результат запроса с сортировкой меток

TopCount (<Множество>, n, <Выражение>) выделяет из множества первые *n* компонент с наибольшими значениями выражения, например запрос

```
Select {[Measures].[Количество],[Measures].[Стоимость]} on Columns,
  TopCount([Продавцы].[Продавец].[Продавец].members, 5,
[Measures].[Стоимость]) on rows
From [Торговля]
```

выберет 5 первых продавцов с наибольшей стоимостью продаж.

Аналогичными функциями являются TopPercent, BottomCount, BottomPercent. Выражение

```
TopPercent([Продавцы].[Продавец].[Продавец].members, 80,
[Measures].[Стоимость])
```

выбирает продавцов, суммарная стоимость продаж которых не меньше 80 %.

Для указания иерархии измерения используются следующие методы и функции:

.children, .FirstChild, . LastChild – «потомки» метки иерархии;

.Parent – «родительская» метка;

.Siblings , .FirstSibling, .LastSibling – «соседи» по уровню.

Перечисленные методы указываются для определенной метки:

[Договоры].[Дата продажи].[Год].&[1996].&[10].parent – родитель для октября 1996г – 1996 год,

[Договоры].[Дата продажи].[Год].&[1996].&[10].FirstChild – первая дата продаж в октябре 1996,

[Договоры].[Дата продажи].[Год].&[1996].&[10].LastChild – последняя дата продаж в октябре 1996,

[Договоры].[Дата продажи].[Год].&[1996].&[10].FirstSibling – первый месяц 1996 года

[Договоры].[Дата продажи].[Год].&[1996].&[10].LastSibling – последний (двенадцатый) месяц 1996 года

Есть специальные функции для формирования множества меток выше по уровню иерархии и ниже уровня по иерархии:

Ascendants(<метка>) – функция возвращает родительскую метку;

Descendants(<метка> [, «Level»[, «Desc_flags»]]) – функция, возвращающая метки, расположенные ниже по уровню иерархии метки – аргумента, например, запрос

```
Select    {[Measures].[Количество],[Measures].[Стоимость]} on Columns,
    {
    Ascendants([Договоры].[Дата продажи].[Год].&[1996].&[10]),
    Descendants([Договоры].[Дата продажи].[Год].&[1996].&[10])
    }  on rows
From     [Торговля]
```

возвращает (Рис. 60) для октября 1996 метки: 10, 1996, All – расположенные выше по иерархии и метки: 3, 11, 19, 27 – расположенные ниже (даты продаж в октябре 1996).

	Количество	Стоимость
10	952	138791,1
1996	16920	3185250
All	35556	6803042
10	952	138791,1
3	203	31773,73
11	346	38490,69
19	193	38861,48
27	210	29665,23

Рис. 60. Результат запроса со списком меток выше и ниже по иерархии

Определение *NON EMPTY <ось>* исключает строки, у которых все клетки пустые (например, не было продаж).

Во всех выражениях языка MDX значения из куба указываются при помощи кортежей. В примере

```
Select{[Measures].[Количество],[Measures].[Стоимость]} on Columns,
    Filter([Товары].[КлассификацияТоваров].[Товар].members,
    ([Measures].[Количество], [Договоры].[Год].&[1996]) >
    ([Measures].[Количество], [Договоры].[Год].&[1997]))  on rows
From [Торговля]
WHERE [Договоры].[Год].&[1997]
```

кортеж используется внутри функции фильтрации, чтобы для каждой метки измерения «Товар» вычислить показатель «Количество» за определенный год.

В запросах можно определять вычисления при помощи следующей конструкции: *WITH <формула> [, <формула>] <Запрос>*

Наиболее часто применяется формулы вычисления уровней измерений
MEMBER <имя уровня> AS '<выражение>'
[,<свойство>=<значение>...]
< имя уровня > – полностью квалифицированное имя с указанием измерения и уровня иерархии, к которому будет отнесено вычисляемое значение.
< выражение > – выражение, вычисляющее значение,

В качестве свойств ячейки можно указывать шрифты и другие особенности форматирования, например, FORMAT_STRING = '# ##0' означает вывод чисел с отбрасыванием дробной части и разделением групп разрядов.

В следующем запросе для каждого периода времени вычисляется суммарная стоимость продаж в сопоставлении с предыдущим периодом (Рис. 61).

```
WITH MEMBER [Measures].[Стоимость товаров за прошлый период]
  AS '([Measures].[Стоимость],   [Договоры].[Дата продажи].PrevMember)'
  MEMBER [Measures].[Увеличение стоимости товаров]
  AS '[Measures].[Стоимость]- [Measures].[Стоимость товаров за прошлый период]'
select {[Договоры].[Дата продажи].[Год] .Members} ON ROWS,
{[Measures].[Стоимость] ,
[Measures].[Стоимость товаров за прошлый период], [Measures].[Увеличение стоимости товаров] } ON COLUMNS
from [Торговля]
```

	Стоимость	Стоимость товаров за прошлый период	Увеличение стоимости товаров
1996	3185250	(null)	3185250
1997	3137318	3185250	-47932
1998	480476,5	3137318	-2656841

Рис. 61. Результат запроса с вычислением новых показателей

Можно вычислять не только новые уровни измерений, но и множества *WITH SET <set_name> AS '<set>'*. Например (Рис. 62):

```
WITH
  SET [Отсортированные товары] AS
    'ORDER( {[Товары].[КлассификацияТоваров].members} ,
                [Measures].[Стоимость], desc)'
  MEMBER [Measures].[All] AS
    '  Sum([Товары].[КлассификацияТоваров].[Товар].Members ,
           [Measures].[Стоимость]) '
  MEMBER [Measures].[Процент] AS
    '  [Measures].[Стоимость] / [Measures].[All] ',
      FORMAT_STRING = '# ##0.0 %'
select { [Measures].[Стоимость],
 [Measures].[Количество],[Measures].[Средняя цена] ,[Measures].[All],
 [Measures].[Процент]}  on columns,
[Отсортированные товары]  on rows
 from [Торговля]
```

	Стоимость	Количество	Средняя цена	All	Процент
All	6803042	35556	191,333164585443	6803043	100,0 %
Крупы	5250831	15832	331,659360788277	6803043,60546875	77,2 %
Гречка	970373,1	2801	346,438080149946	6803043,60546875	14,3 %
Рис	931824,3	2736	340,57903874269	6803043,60546875	13,7 %
Пшено	905662,6	3295	274,859673748103	6803043,60546875	13,3 %
Овсянка	887262,8	2337	379,658884253316	6803043,60546875	13,0 %
Крупа манная	848554,9	2040	415,958302696078	6803043,60546875	12,5 %
Крупа перловая	707153,3	2623	269,59712161647	6803043,60546875	10,4 %
Табачные изделия	1197066	9920	120,672001008065	6803043,60546875	17,6 %
Сигареты MarlBoro	398807,4	2622	152,100471967963	6803043,60546875	5,9 %
Сигареты Fine Line	322485,7	2453	131,465819914391	6803043,60546875	4,7 %
Сигареты LM	300945,2	2485	121,104715794769	6803043,60546875	4,4 %
Сигареты Петр 1	174827,8	2360	74,0795881885593	6803043,60546875	2,6 %
Кондитерские изделия	355146,5	9804	36,2246563902489	6803043,60546875	5,2 %
Конфеты "Визит"	151652,8	2548	59,5183722527473	6803043,60546875	2,2 %
Конфеты "Асорти"	77432,13	2489	31,1097328244275	6803043,60546875	1,1 %
Конфеты "Ананасные"	71624,13	1840	38,9261591372283	6803043,60546875	1,1 %
Конфеты "Весна"	54437,46	2927	18,5983795802016	6803043,60546875	0,8 %

Рис. 62. Результат запроса с вычислением множества

MDX-запросы можно использовать для извлечения данных из кубов аналогично SQL-запросам. Такое использование обладает большими аналитическими возможностями и высоким быстродействием.

Вопросы

1. Назначение аналитических баз данных. Отличие хранилищ аналитических баз от реляционных баз данных.
2. Причины построения и использования аналитических баз данных.
3. Требования, предъявляемые к аналитическим базам данных.
4. Опишите структуру данных аналитической базы.
5. Дайте определение понятий «куб», «измерение», «показатель», «ячейка куба», «метка измерения».
6. Перечислите и опишите основные операции в кубе.
7. Опишите технологию создания куба с помощью SQL Server Data Tools.
8. Что содержит таблица фактов?
9. Для чего используются обобщения (агрегаты)?
10. Перечислите и опишите режимы хранения данных в кубе.
11. Как обновляется содержание куба?
12. Опишите систему безопасности MS Analysis Services.
13. Какие средства можно использовать для доступа к данным аналитической базы?
14. Для чего предназначен язык MDX? Каковы его основные возможности?
15. Как выглядит запрос на языке MDX? Каковы его основные компоненты?
16. Описание кортежей, множеств, меток в языке MDX.

17. Опишите применение функций сортировки, фильтрации, комбинирования измерений в языке MDX.

18. Опишите применение функций работы с иерархическими структурами в языке MDX.

19. Укажите возможности программирования вычислений в языке MDX.

8. Нереляционная база данных MongoDB

8.1. Структуры данных MongoDB

Реляционные БД отличаются большой «жесткостью». В реляционную БД можно записать данные, только если есть соответствующие таблицы и поля. Любые дополнительные сведения требуют изменения схемы реляционной БД. Другой особенностью реляционной БД является описание единого документа связанными записями из разных таблиц. Для описания договора сведения о продавце и покупателе записываются в одноименные таблицы. Список продаж договора помещается в таблицу, содержащую списки продаж всех договоров. Для сбора всех данных выполняются многочисленные соединения таблиц оператором JOIN.

База данных MongoDB [1; 9] предлагает альтернативный способ организации, хранения и обработки данных, ориентированный на описание документов и других данных единым блоком произвольной структуры. MongoDB относят к документо-ориентированным базам данных. Все «новые» базы данных, не поддерживающие реляционные принципы организации данных, объединяют в множество NoSQL баз данных. NoSQL (от англ. not only SQL – не только SQL) – термин, обозначающий ряд подходов, направленных на реализацию систем управления базами данных, имеющих существенные отличия от моделей, используемых в традиционных реляционных СУБД с доступом к данным средствами языка SQL (https://ru.wikipedia.org/wiki/NoSQL).

Еще одной особенностью MongoDB является ее распределенность. Система хранения данных в MongoDB представляет набор реплик. В этом наборе есть основной узел, а также может быть набор вторичных узлов. Все вторичные узлы сохраняют целостность и автоматически обновляются вместе с обновлением главного узла. И если основной узел по каким-то причинам выходит из строя, то один из вторичных узлов становится главным.

Базовой структурой данных является документ в формате BSON (БиСон) – сокращение от binary JSON (JavaScript Object Notation). В этом формате описание товара выглядит следующим образом:

```
{
    "Код товара" : 14,
    "Товар" : "Сигареты Петр 1",
    "Единица измерения" : "Блок",
    "Вес ЕдИзм(Кг)" : 0.2,
    "Группа товаров" : "Табачные изделия"
}
```

Каждое поле описывается именем и значением. Тип значения определяется по записи значения, например, 564781 – целое число, "564781" – строка. Возможно использование следующих типов данных (https://proselyte.net/tutorials/mongodb/data_types/):

- Integer – целочисленные значения;
- Double – значения с плавающей точкой;

– Boolean – логические (true / false) значения;
– String – строки символов в кодировке UTF-8;
– Arrays – массивы;
– Object – документы;
– Date – даты в UNIX-формате.

Таким образом, каждый документ может иметь любую структуру, которая описывается прямо в документе. Документы «одного типа» объединяют в коллекцию. Понятие «одного типа» весьма условно. В одну коллекцию можно поместить документы любых структур. Однако эти возможности все равно следует ограничить здравым смыслом. Для того чтобы выбор по полям имел смысл, поле выбора должно быть в большинстве документов коллекции. Только в этом случае можно организовать массовую обработку. Поэтому документы в одной коллекции должны иметь приблизительно одинаковую структуру и одинаковое назначение.

Итак, коллекция объединяет информационные объекты близки по назначению и способам обработки, но каждый из них может иметь специфическое описание. Например, для конфет можно ввести срок годности:

```
{
    "Код товара" : 2,
    "Товар" : "Конфеты \"Ананасные\"",
    "Единица измерения" : "Коробка",
    "Вес ЕдИзм(Кг)" : 0.3,
    "Группа товаров" : "Кондитерские изделия",
    "Срок годности" : "22.12.1996"
}
```

Чем больше будут отличаться информационные объекты в коллекции, тем сложнее будет определять универсальную обработку всех документов. Например, для выбора продаж товаров с не истекшим сроком годности, нужно проверять наличие такого поля, и, если оно есть, то проверять его значение.

База данных в MongoDB объединяет несколько коллекций для адекватного описания некоторой предметной области.

Для каждого документа в MongoDB определен уникальный идентификатор, который называется id. При добавлении документа в коллекцию данный идентификатор создается автоматически. Однако разработчик может сам явным образом задать идентификатор, а не полагаться на автоматически генерируемые, указав соответствующий ключ и его значение в документе.

Если идентификатор не задан явно, то MongoDB создает специальное бинарное значение размером 12 байт. Это значение состоит из нескольких сегментов: значение типа timestamp размером 4 байта, идентификатор машины из 3 байт, идентификатор процесса из 2 байт и счетчик из 3 байт. Таким образом, первые 9 байт гарантируют уникальность среди других машин, на которых могут быть реплики базы данных. А следующие 3 байта гарантируют уникальность в течение одной секунды для одного процесса. Такая модель построения идентифика-

тора гарантирует с высокой долей вероятности, что он будет иметь уникальное значение.

Рассмотрим, как можно было бы хранить в MongoDB базу данных со схемой с Рис. 12 с таблицами «Продавцы», «Покупатели», «Договоры», «ТоварыВдоговорах», «Товары». Можно было бы ограничится одной коллекцией «Договоры», включив в нее описание и продавца, и покупателя, и массив продаж с описанием описание товара в каждой продаже. Игнорирование избыточности характерно для MongoDB. Однако просто неудобно извлекать из такой коллекции список покупателей, или продавцов, или товаров, тем более что часто нужно описать товар, даже если не было ни одной продажи. Поэтому создадим отдельные коллекции «Продавцы», «Покупатели», «Товары», а договор будет описываться одним документов с вложенным массивом продаж:

```
{
   "Номер договора" : 45,
   "Дата" : "22.12.1996",
   "Код продавца" : 8,
   "Код покупателя" : 3,
   "Предоплата" : 0.5,
   "Продажи" : [
         {
               "Код товара" : 1,
               "Количество" : 13,
               "Цена" : 427
         },
         …
         {
               "Код товара" : 13,
               "Количество" : 6,
               "Цена" : 220
         }
   ]
}
```

Полученная схема частично сохраняет реляционную идеологию: договор содержит ссылки в виде значений соответствующих ключей для указания продавца, покупателя и товаров в соответствующих коллекциях. Включение в договор массива продаж, наоборот, вполне в духе MongoDB: описание объекта задается одним документом.

8.2. Дистрибутив MongoDB

Пакет MongoDB можно загрузить с официального сайта https://www.mongodb.com/download-center/community. После установки становятся доступны следующие программы:

– **bsondump**: считывает содержимое BSON-файлов и преобразует их в читабельный формат, например, в JSON;

– **mongo**: представляет консольный интерфейс для взаимодействия с базами данных, своего рода консольный клиент;

– **mongod**: сервер баз данных MongoDB. Он обрабатывает запросы, управляет форматом данных и выполняет различные операции в фоновом режиме по управлению базами данных;

– **mongodump**: утилита создания бэкапа баз данных, команды с cd C:\Program Files\MongoDB\Server\4.0\bin mongodump --db=Sales --out=d создают дамп базы Sales на диске d;

– **mongoexport**: утилита для экспорта данных в форматы JSON, TSV или CSV;

– **mongofiles**: утилита, позволяющая управлять файлами в распределенной системе GridFS;

– **mongoimport**: утилита, импорирующая данных в форматах JSON, TSV или CSV в базу данных MongoDB;

– **mongorestore**: позволяет записывать данные из дампа, созданного mongodump, в новую или существующую базу данных (**mongorestore /dir:d:\Sales /db:SalesR)**;

– **mongos**: служба маршрутизации MongoDB, которая помогает обрабатывать запросы и определять местоположение данных в кластере MongoDB;

– **mongorestat**: представляет счетчики операций с БД;

– **mongotop**: предоставляет способ подсчета времени, затраченного на операции чтения-записи в БД.

Сервер представляет приложение **mongod**, которое находится в папке **bin**. Запуск приложения **mongod** приводит к запуску служб сервера. Параметры сервера определяются содержимым файла **mongod.cfg**. В ОС Windows по умолчанию MongoDB хранит базы данных по пути C:\data\db. Для использования другой папки ее нужно указать при запуске MongoDB в параметре --dbpath.

После запуска сервера можно производить операции с БД через оболочку mongo. Эта оболочка представляет файл mongo.exe, который располагается в папке установки **bin**. Чтобы использовать mongoDB с приложениями на PHP, C++, C# и других языках программирования потребуются специальные драйверы. На сайте на странице https://docs.mongodb.com/ecosystem/drivers/ можно найти драйверы для таких языков программирования, как PHP, C++, C#, Java, Python, Perl, Ruby, Scala и др.

8.3. Работа с базой данных MongoDB

Подключившись к серверу через оболочку **mongo**, можно передавать серверу различные команды. Завершение оболочки указывается командой quit() или комбинацией клавиш Ctrl-C. Кроме этого стандартного редактора команд, разработаны и применяются командные оболочки с графическим интерфейсом. Свободно распространяемая оболочка **MongoDB Compass** предоставляет средства просмотра и редактирования документов в коллекциях, а также инструмент выбора данных с фильтрацией и агрегированием, однако в ней отсутству-

ет возможность запуска команд и скриптов команд. Поэтому все примеры будут демонстрироваться в **mongo**.

После запуска mongo открывается окно команд с приглашением «>». Приведем некоторые команды сервера:

- получить список БД: show dbs;
- использовать базу данных, указанную именем: use <имя базы данных>;
- получить справку по методам БД: db.help();
- получить статистику по БД: db.stats();
- удаление текущей БД: db.dropDatabase();
- получить список коллекций текущей БД: db.getCollectionNames();
- создать коллекцию: db.createCollection(<имя коллекции>);
- удалить коллекцию: db. <имя коллекции>.drop();
- получить список методов коллекции: db. <имя коллекции>.help().

Для создания БД, коллекции и документа можно выполнить запись в новую (еще не существующую) базу нового документа в новую коллекцию:

```
use myNewDB
db.myNewCollection1.insertOne( { x: 1 } )
```

Всегда работает правило: если указанный в команде добавления объект не существует, то он создается. После указанных команд на сервере появится БД myNewDB с коллекцией myNewCollection1 и новым документом { x: 1 } в ней. Для базы данных можно задать любое имя, однако есть некоторые ограничения. Например, в имени не должно быть символов /, \, ., ", *, <, >, :, |, ?, $. Кроме того, имена баз данных ограничены 64 байтами. Также есть зарезервированные имена, которые нельзя использовать: local, admin, config.

8.4. Команды CRUD изменения документов коллекции

Стандартные операции по изменению БД – добавить документ, удалить документ, изменить документ – выглядят и определяются как соответствующие методы коллекции.

Для добавления в коллекцию могут использоваться три ее метода:

- insertOne(): добавляет один документ;
- insertMany(): добавляет несколько документов;
- insert(): может добавлять как один, так и несколько документов.

Например, команда

```
db.Товары.insert(
   {"Код товара":15,
   "Товар":"Конфеты \"Белочка\"",
   "Единица измерения":"Коробка",
   "Вес ЕдИзм(Кг)":0.5,
   "Группа товаров":"Кондитерские изделия",
   "Срок годности" : "22.12.1996"}
   )
```

добавит в коллекцию «Товары» один документ. Если не задано значение поля "_id" (уникального идентификатора документа), то оно генерируется автоматически.

Для удаления из коллекции используется метод **remove**({<условие удаления>}). Удаляются все документы коллекции, удовлетворяющие условию. Например, **db.Товары.remove ({"Код товара":15}).** Если условие не задано, удаляются все документы коллекции.

Использование индексов

Индексы позволяют значительно повысить скорость поиска по значению индекса. Кроме этого, часто нужно обеспечить уникальность индекса для исключения возможности появления дубликатов.

Например, можно создать индекс по полю "Код товара"

db.Товары.createIndex({"Код товара" : 1}, {"unique" : true})

Для просмотра списка индексов используется метод getIndexes(). Удалить индекс можно методом dropIndex, например, db.Товары.dropIndex("Код товара_1").

В использовании индексов есть особенности. Так, документ может не содержать индексируемое поле ("Код товара"). В этом случае для добавляемого документа автоматически создается ключ "Код товара" со значением null. Поэтому при добавлении второго документа, в котором не определен ключ "Код товара" будет приводить к ошибке – появление дублирующего значения.

Также можно задать уникальный индекс сразу для двух полей:

Обновление данных

Метод save

Наиболее простым для использования является метод save. В качестве параметра этот метод принимает документ. В этот документ в качестве поля можно передать параметр _id. Если метод находит документ с таким значением _id, то документ обновляется. Если же с подобным _id нет документов, то документ вставляется. Если параметр _id не указан, то документ вставляется, а параметр _id генерируется автоматически как при обычном добавлении через функцию insert:

```
db.Товары.save({
    "_id" : ObjectId("5d196ec9393c7f23ac0bbffe"),
    "Код товара" : 15,
    "Товар" : "Конфеты \"Белочка\"",
    "Единица измерения" : "Коробка",
    "Вес ЕдИзм(Кг)" : 0.5,
    "Группа товаров" : "Кондитерские изделия",
    "Срок годности" : "22.12.1996"
})
```

В качестве результата функция возвращает объект WriteResult. Например, при успешном сохранении мы получим:

WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })

Метод update

Более детальную настройку при обновлении предлагает функция update. Она принимает три параметра:

– query: принимает запрос на выборку документа, который надо обновить;

– objNew: представляет документ с новой информацией, который заместит старый при обновлении;

– options: определяет дополнительные параметры: upsert и multi.

Если upsert=true, что mongodb будет обновлять документ, если он найден, и создавать новый, если такого документа нет. Если upsert=false, то mongodb не будет создавать новый документ, если запрос на выборку не найдет ни одного документа.

Параметр multi указывает, должен ли обновляться первый элемент в выборке (используется по умолчанию, если данный параметр не указан) или же должны обновляться все документы в выборке.

Например:

```
db.Товары. update (
{"Код товара" : 15},
{
   "Код товара" : 15,
   "Товар" : "Конфеты \"Белочка\"",
   "Единица измерения" : "Коробка",
   "Вес ЕдИзм(Кг)" : 0.5,
   "Группа товаров" : "Кондитерские изделия",
   "Срок годности" : "22.12.1996"
},
{upsert: true})
```

Будет перезаписан документ о товаре с кодом 15.

Обновление отдельного поля

Для изменения отдельных полей используется оператор $set. Если документ не содержит обновляемое поле, то оно создается.

```
db.Товары. update ({"Код товара" : 15}, {$set: {"Срок годности" : 30}})
```

В данном случае обновлялся только один документ, первый в выборке. Указав значение multi:true, мы можем обновить все документы выборки:

```
db.Товары. update ({"Группа товаров" : "Кондитерские изделия"}, {$set: {"Срок годности" : 30}}, {multi:true})
```

Удаление поля

Для удаления отдельного ключа используется оператор $unset:

```
db.Товары. update ({"Группа товаров" : "Кондитерские изделия"}, {$unset: {"Срок годности" : 1}}, {multi:true})
```

Если вдруг подобного ключа в документе не существует, то оператор не оказывает никакого влияния.

Обновление массивов

Оператор $push позволяет добавить один элемент в конец массива. Например,

```
db.Договоры.update({"Номер договора" : 45},
{$push: {"Продажи":
        {
            "Код товара" : 2,
            "Количество" : 13,
            "Цена" : 60
        }
}})
```

Оператор $position задает позицию в массиве для вставки элементов.

Оператор $addToSet подобно оператору $push добавляет объекты, если их еще нет в массиве:

Удаление элемента из массива

Оператор $pop позволяет удалять последний элемент массива:

```
db.Договоры.update({"Номер договора" : 45},{$pop: {"Продажи": 1}})
```

Чтобы удалить первый элемент сначала массива, надо передать отрицательное значение:

```
db.Договоры.update({"Номер договора" : 45},{$pop: {"Продажи": -1}})
```

Оператор $pull удаляет каждое вхождение элемента в массив. Например,

```
db.Договоры.update({"Номер договора" : 45},
{$pull : {"Продажи":
{
            "Код товара" : 2,
            "Количество" : 13,
            "Цена" : 60

}}})
```

8.5. Выборка из БД

Выборка документов из коллекции выполняется с помощью метода find. Например, чтобы извлечь все документы из коллекции «Товары» можно использовать команду db. Товары.find().

Для вывода документов в наглядном представлении можно добавить вызов метода pretty():db. Товары.find().pretty().

Условие выбора является аргументом метода find, например, команда

```
db.Товары.find({"Группа товаров" : "Кондитерские изделия", "Вес ЕдИзм(Кг)" : 0.5})
```

выберет товары с заданными значениями двух полей.

Условные операторы

Условные операторы задают условие, которому должно соответствовать значение поля документа:

– $eq (равно);
– $ne (не равно);
– $gt (больше чем);
– $lt (меньше чем);
– $gte (больше или равно);
– $lte (меньше или равно);
– $in определяет массив значений, одно из которых должно иметь поле документа;
– $nin определяет массив значений, которые не должно иметь поле документа.

Например, найдем все товары, у которых "Вес ЕдИзм(Кг)" меньше 0.5:

db.Товары.find({"Вес ЕдИзм(Кг)" : {$lt : 0.5}}). Команда

db.Товары.find({"Вес ЕдИзм(Кг)" : {$gt : 0.2, $lt : 0.5}})

выберет все товары, у которых "Вес ЕдИзм(Кг)" больше 0.2 меньше 0.5

Оператор $in определяет массив возможных выражений и ищет те значения, которые есть в массиве:

db.Товары.find({"Код товара" : {$in : [2, 5, 9]}})

Логические операторы

Логические операторы выполняются над условиями выборки:
– $or: документ должен соответствовать одному из этих условий;
– $and: документ должен соответствовать обоим условиям;
– $not: документ должен НЕ соответствовать условию;
– $nor: должен не соответствовать условиям.

Например, команда

db.Товары.find({ $or: [{"Группа товаров" : "Кондитерские изделия"}, {"Вес ЕдИзм(Кг)": {$gt: 10}}]})

выбирает товары группы "Кондитерские изделия" или товары с весом единицы больше 10.

Оператор $exists

Оператор $exists позволяет извлечь только те документы, в которых поле присутствует или отсутствует. Например, команда

db.Товары.find({"Срок годности" : {$exists: true}})

выбирает документы, в которых указано поле "Срок годности".

Оператор $type

В документе одноименные поля могут иметь разный тип. Оператор $type извлекает только те документы, в которых определенный ключ имеет значение определенного типа, например, строку или число:

db.Товары.find({"Вес ЕдИзм(Кг)" : {$type:"number"}})

Оператор $regex

Оператор $regex

{ <поле>: { $regex: 'образец', $options: '<опции>' } }

задает регулярное выражение, которому должно соответствовать значение поля. Например, пусть поле "Товар" обязательно имеет слово "конфеты":

db.Товары.find({"Товар": {$regex:"конфеты", $options: 'i'}}).

Параметр $options: 'i' означает игнорирование строчного и заглавного регистра.

Поля вложенных документов

Если документ содержит вложенную структуру, то для указания ее полей используют имя структуры и поля, разделенные точкой. Например, если при описании продавца используется структура

```
"Адрес" : {
        "Улица" : "Партизанская",
        "Дом" : "37а",
        "Офис" : "45"
},
```

то команда db.Продавцы.find({ "Адрес.Улица": "Партизанская" }) выполнит поиск по улице.

Поиск по массивам

Ряд операторов предназначены для работы с массивами:

– $all: определяет набор значений, которые должны иметься в массиве;

– $size: определяет количество элементов, которые должны быть в массиве;

– $elemMatch: определяет условие, которым должны соответствовать элементы в массиве.

Оператор $size проверяет количество элементов массива:

db.Договоры.find({"Продажи" :{$size: 8}})

Оператор $elemMatch позволяет выбрать документы, в которых массивы содержат элементы, попадающие под определенные условия. Команда

db.Договоры.find({"Продажи": {$elemMatch: {"Код товара": 7}}})

выбирает договоры, у которых в списке продаж есть товар с кодом 7.

Проекция

Инструмент «проекция» позволяет выбирать отдельные поля. Это второй аргумент в методе find. Можно просто для каждого поля указать включение и исключение.

Команда

db.Договоры.find({"Продажи": {$elemMatch: {"Код товара": 7}}}, {"_id":0, "Номер договора" :1, "Дата" :1, "Код продавца":1})

определяет исключение поля "_id" и выбор полей "Номер договора", "Дата", "Код продавца".

Настройка запросов

MongoDB представляет ряд свойств, которые помогают управлять выборкой из БД. Свойство limit задает максимально допустимое количество получаемых документов. Количество передается в виде числового параметра. Например, ограничим выборку тремя документами:

db.Продавцы.find({}).limit(3)

Свойство skip позволяет пропускать записи:

db.Продавцы.find({}).limit(3).skip(3)

Свойство sort управляет сортировкой выбираемых документов. Указание для поля 1 или -1 устанавливает порядок сортировки: по возрастанию или по убыванию. Например,

db.Продавцы.find({}).limit(3).skip(3).sort({"Город":-1})

Параметр sort({ $natural: -1 }) позволяет использовать в сортировке порядок добавления в коллекцию.

Свойство count() вычисляет количество документов в запросе:

db.Договоры.find({"Продажи": {$elemMatch: {"Код товара": 7}}}).count()

Команды группировки

Использование метода group аналогично применению выражения GROUP BY в SQL. Метод group принимает обязательные параметры:

– key: поле, по которому надо проводить группировку;

– initial: инициализирует поля документа, который будет описывать группу документов;

– reduce: функция, вычисляющая агрегированные по группе значения с двумя аргументами: текущий элемент и агрегатный результирующий документ для текущей группы,

и необязательные параметры:

– keyf: используется вместо параметра key и представляет функцию, которая возвращает объект key;

– cond: условие, по которому выбираются документы в группу;

– finalize: функция, которая срабатывает перед тем, как будут возвращены результаты группировки.

Команда

```
db. Договоры.group ({
    key: { "Код продавца" : true},
    initial: {Договоров : 0},
    reduce : function (curr, res){res.Договоров += 1}
})
```

для каждого продавца вычисляет количество договоров.

Применение агрегирования

Метод db.<коллекция>.aggregate([{ <стадия> }, ...]) является мощным инструментом вычисления нужных итогов по коллекции документов. Метод использует конвейерную обработку документов, выполняя по очереди стадии обработки (Aggregation Pipeline Stages. URL: https://docs.mongodb.com/manual/reference/operator/aggregation-pipeline/). Входом для каждой стадии является выход предыдущей. Для первой стадии входом является коллекция, к которой применяется агрегирование. Последняя стадия формирует результат работы команды.

Стадия { $match: { <условия> } } реализует выборку аналогично фразе WHERE в SQL. Если это первая стадия, то выборка выполняется из агрегируемой коллекции и ее результатом будут выбранные документы.

Стадия

```
{ $group: { _id: <выражение>,
            <field1>: { <агрегация1> : <выражение1> },
            ... } }
```

выполняет группировку документов и вычисления по группе. В одну группу попадают документы с одинаковым значением выражения. По группе вычисляются агрегированные значения с помощью функций агрегирования:

- $addToSet – формирует массив уникальных значений выражений;
- $avg – вычисляет среднее значение выражений;
- $first – возвращает первое выражение в группе;
- $last – возвращает последнее выражение в группе;
- $max – вычисляет максимальное значение среди выражений группы;
- $mergeObjects – возвращает комбинацию полей выражений;
- $min – вычисляет минимальное значение среди выражений группы;
- $push – формирует массив из значений выражений группы;
- $stdDevPop – вычисляет генеральное среднеквадратическое отклонение;
- $stdDevSamp – вычисляет выборочное среднеквадратическое отклонение;
- $sum – вычисляет сумму выражений.

При вычислении числовых агрегированных значений нечисловые значения игнорируются.

Стадия

```
{
  $lookup:
    {
     from: < коллекция поиска>,
     localField: <поле-ссылка на ключ>,
     foreignField: <ключ в коллекции поиска>,
     as: <поле для найденных документов>
    }
}
```

реализует поиск документов в указанной коллекции по значению поля-ссылки в текущей коллекции. Возвращает массив найденных документов. Данная стадия аналогична левому внешнему соединению в SQL.

Стадия

```
{ $project: { <определение поля>, … } }
```

для каждого документа входящей коллекции формирует документ из определенного списка полей. Определение поля может быть одним из следующих:

<поле>: {0|1} – включение или исключение поля входного документа в выходной;

<поле>: <выражение> – вычисление значения выходного документа.

Для вычисления выражений используются специальные операторы [Aggregation Pipeline Quick Reference. URL: https://docs.mongodb.com/manual/meta/aggregation-quick-reference/#operator-expressions], в которых для указания поля перед его обозначение указывается символ $.

Стадия

{ $sort: { <поле1>: <порядок1>, <поле2>: <порядок2>, ... } }

выполняет сортировку по указанным полям входных документов. Порядок сортировки указывается значениями: 1 – по возрастанию, –1 – по убыванию.

В команде

```
db.Договоры.aggregate([
  { $match: { "Код покупателя" : 3 } },
  { $group: { _id: "$Код продавца",
        "СредняяПредоплата": { $avg: "$Предоплата" } } },
  {$lookup: {
     from: "Продавцы",
     localField: "_id",
     foreignField: "Код продавца",
     as: "ОписаниеПродавца"
    }},
  {$project: { _id:0, "ОписаниеПродавца.Продавец":1, "Средняя предоплата
        %": { $multiply: [ "$СредняяПредоплата", 100 ] } } },
  { $sort: {"Средняя предоплата %":-1 } }
])
```

первая стадия выбирает из коллекции «Договоры» документы, удовлетворяющие условию "Код покупателя" : 3. Вторя стадия $group выполняет группировку отфильтрованных документов по коду продавца и формирует коллекцию документов из двух полей: "Код продавца" и "СредняяПредоплата". Обратите внимание на употребление знака $ для указания использования не строки, а имени поля. Третья стадия $lookup добавляет к результату группировки поле "ОписаниеПродавца", которое содержит массив из одного документа, описывающего найденного по коду продавца. Ниже приведен пример выбранного этой стадией документа:

```
{
  "_id" : 8,
  "СредняяПредоплата" : 0.5,
  "ОписаниеПродавца" : [
        {
              "_id" : ObjectId("5cf9d27adcc8fe5a26fa61be"),
              "Код продавца" : 8,
              "Форма собственности" : "АООТ",
              "Продавец" : "Гермес",
              "Город" : "Ангарск",
              "Банк" : "Богатырский"
        }
  ]
}
```

Четвертая стадия $project выбирает документы с двумя полями "ОписаниеПродавца.Продавец", "Средняя предоплата %". Второе поле вычисляется умножением на 100. Последняя, пятая, стадия выполняет сортировку по убыванию средней предоплаты. Пример документа, полученного командой, приведен ниже:

```
    {
      "ОписаниеПродавца" : [
            {
                  "Продавец" : "Гермес"
            }
      ],
      "Средняя предоплата %" : 50
    },
```

Обратите внимание на сохранение проекцией структуры: описание продавца по-прежнему представлено массивом. Для извлечения поля из массива можно воспользоваться оператором $arrayElemAt:

```
{$project: { _id:0,
    "Продавец": { $arrayElemAt: ["$ОписаниеПродавца.Продавец", 0 ] },
    "Средняя предоплата %": { $multiply: [ "$СредняяПредоплата", 100 ] }
}}
```

В этом случае выходной документ будет содержать два поля.

Команда

```
db.Договоры.aggregate([
  {$project: {
     _id:0,
     "Номер договора" : 1,
     "СтоимостьДоговора":{
       $reduce: {
         input: "$Продажи",
         initialValue: 0 ,
         in: {
           $add : ["$$value", {$multiply: ["$$this.Количество", "$$this.Цена" ]]}}
         }
     }
  }
])
```

демонстрирует вычисление по массиву продаж суммарной стоимости договора с помощью оператора $reduce. В этом операторе

```
{
  $reduce: {
     input: <массив>,
     initialValue: <выражение>,
     in: <выражение>
  }
}
```

параметр initialValue определяет начальное значение внутренней переменной $$value, которая после просмотра всех компонентов массива будет содержать вычисленное значение. Параметр in определяет правило пересчета $$value для каждого компонента массива. Текущий компонент массива обозначается как $$this. В рассмотренном выше примере для каждого компонента массива вычисляется стоимость продажи умножением цены на количество, и вычисленная стоимость добавляется к уже накопленной в $$value сумме.

Если необходимо проводить вычисления по товарам, то следует преобразовать договоры с вложенными массивами продаж в коллекцию продаж, т.е. каждый компонент массива должен быть преобразован в документ. Для этого можно использовать стадию $unwind:

{ $unwind: <поле> }

Указанное поле должно быть массивом. В результате выполнения команды каждый компонент массива преобразуется в отдельный документ, который будет содержать поля документа массива и поля компонента массива.

Например, для команды

db.Договоры.aggregate({$unwind: "$Продажи"})

ниже приведен фрагмент результата


```
    {
        "_id" : ObjectId("5cf9e533dcc8fe5a26fa66fb"),
        "Номер договора" : 44,
        "Дата" : "14.12.1996",
        "Код продавца" : 5,
        "Код покупателя" : 6,
        "Предоплата" : 0.4,
        "Продажи" : {
                "Код товара" : 13,
                "Количество" : 70,
                "Цена" : 191
        }
    },
    {
        "_id" : ObjectId("5cf9e533dcc8fe5a26fa66fb"),
        "Номер договора" : 44,
        "Дата" : "14.12.1996",
        "Код продавца" : 5,
        "Код покупателя" : 6,
        "Предоплата" : 0.4,
        "Продажи" : {
                "Код товара" : 1,
                "Количество" : 23,
                "Цена" : 455
        }
    },
```

Пример демонстрирует, что каждый договор превращается в несколько по числу компонентов в массиве «Продажи» и каждый новый документ содержит объект «Продажи», описывающий один компонент массива.

Для получения информации о товарах дополним конвейер стадией $lookup

```
db.Договоры.aggregate(
{$unwind: "$Продажи"},
{$lookup: {
      from: "Товары",
      localField: "Продажи.Код товара",
      foreignField: "Код товара",
      as: "ОписаниеТовара"
     }},
)
```

В результате получается набор документов вида

```
{
      "_id" : ObjectId("5cf9e533dcc8fe5a26fa66fb"),
      "Номер договора" : 44,
      "Дата" : "14.12.1996",
      "Код продавца" : 5,
      "Код покупателя" : 6,
      "Предоплата" : 0.4,
      "Продажи" : {
            "Код товара" : 13,
            "Количество" : 70,
            "Цена" : 191
      },
      "ОписаниеТовара" : [
            {
                  "_id" : ObjectId("5cf9d45adcc8fe5a26fa6341"),
                  "Код товара" : 13,
                  "Товар" : "Сигареты MarlBoro",
                  "Единица измерения" : "Блок",
                  "Вес ЕдИзм(Кг)" : 0.2,
                  "Группа товаров" : "Табачные изделия"
            }
      ]
},
```

Теперь можно выполнить сводные вычисления по продажам каждого товара:

```
db.Договоры.aggregate([
  {$unwind: "$Продажи"},
  {$lookup: {
      from: "Товары",
      localField: "Продажи.Код товара",
```

```
        foreignField: "Код товара",
        as: "ОписаниеТовара"
      }},
  {$group: { _id: { $arrayElemAt: ["$ОписаниеТовара.Код товара", 0 ] },
   "Товар": {$first: {$arrayElemAt: ["$ОписаниеТовара.Товар", 0 ]}},
   "Единица измерения":
              {$first: {$arrayElemAt: ["$ОписаниеТовара.Единица измерения", 0 ]}},
   "СуммарноеКоличество": { $sum: "$Продажи.Количество" }  ,
   "СуммарныйВес": { $sum: {
        $multiply: [ { $arrayElemAt: ["$ОписаниеТовара.Вес ЕдИзм(Кг)", 0 ] },
        "$Продажи.Количество" ] } },
   "СуммарнаяСтоимость": { $sum: {
         $multiply: [ "$Продажи.Цена", "$Продажи.Количество" ] } }
  },
  {$project: { _id: 0 } },
  {$sort: {"СуммарнаяСтоимость":-1 } }
])
```

В этой команде приходится применять оператор $arrayElemAt, потому что стадия $lookup всегда возвращает массив (в данной команде в нем всегда будет один элемент). В команде оператор выбирает первый элемент массива с номером 0. Функция агрегирования $first применяется в команде для выбора значений полей «Товар» и «Единица измерения», так как они одинаковые для всех документов группы.

8.6. Распределенная обработка данных Map-Reduce

Распределенная обработка позволяет выполнить сбор и обработку данных, распределенных в нескольких базах данных, размещенных на разных серверах. СУБД MongoDB изначально проектировалась как распределенная система, поэтому она не могла не включить соответствующий механизм. Используется технология Map-Reduce, разработанная Google, и получившая широкое распространение

Технология Map-Reduce заключается в следующем. Вначале к каждому документу коллекции применяется операция Map, которая формирует пары <ключ, значение> с исходной для обработки информацией. Эти пары затем передаются функции reduce в сгруппированном по ключу виде. Операция Reduce формирует по группе одну пару <ключ, значение>. Как в качестве ключа, так и в качестве значения могут выступать любые переменные, включая объекты. Операция Map может выполняться на разных серверах формируя потоки данных, объединяемые операцией Reduce в итоговую коллекцию, содержащую итоги распределенной обработки данных.

Предположим, что необходимо обработать договоры о продажах для получения сводных данных по продажам каждого товара. Следующая функция

```
function map(){
        for(var i in this.Продажи) {
                var c = this.Продажи[i].Количество * this.Продажи[i].Цена;
```

```
            emit(this.Продажи[i].КодТовара,
            {count: 1,  quantity: this.Продажи[i].Количество,  cost: c })
        }
};
```

для каждой продажи формирует пару: КодТовара, кортеж, содержащий показатели продажи
<КодТовара, {"count" : 1,"quantity" : количество, "cost" : стоимость}>.

Документ, к которому применена операция Map, доступен по указателю this. Функция emit служит для передачи очередной пары <ключ, значение> для дальнейшей обработки. Операция Map для одного документа («договор») может выдать несколько пар <ключ, значение>.

Сформированные пары группируются по ключу и передаются функции reduce в виде <ключ, список значений>. Функция reduce для нашего примера выглядит так:

```
function reduce(key, values) {
   var quantity = 0;
        var count = 0;
        var cost = 0;
for(var i in values) {
   count += values[i].count;
   quantity += values[i].quantity;
   cost += values[i].cost;
   }
return {count: count, quantity: quantity, cost: cost };
};
```

MongoDB запускает операцию Reduce для выполнения промежуточных агрегаций. Как только сформировано несколько пар с одинаковым ключом, MongoDB может выполнить для них Reduce, получив тем самым одну пару <ключ, значение>, которая потом обрабатывается наравне с остальными, как если бы она была получена с помощью операции Map. Это накладывает определенные требования на реализацию функции reduce. Вот они:

1. Тип возвращаемого значения функции reduce должен совпадать с типом значения, которое выдается функцией map (второй параметр функции emit).

2. Должно выполняться равенство: reduce(key, [A, reduce(key, [B, C])]) = = reduce(key, [A, B, C]).

3. Повторное применение операции Reduce к полученной паре <ключ, значение> не должно влиять на результат (идемпотентность).

4. Порядок значений, передаваемых функции reduce, не должен влиять на результат.

Чтобы запустить процедуру Map-Reduce, нужно объявить эти две функции, а затем выполнить команду:

```
db.Договоры.mapReduce(map, reduce, {out:"Goods"});
```

СУБД выполнит указанную обработку и вернет результат, фрагмент которого приведен ниже:

```
{
	"1" : {
		"count" : 50,
		"quantity" : 2801,
		"cost" : 970264
	},
	"2" : {
		"count" : 55,
		"quantity" : 1840,
		"cost" : 71536
	},
	…
}
```

Описанная технология позволяет накапливать итоги в виде сумм, количеств в соответствии с требованиями масштабирования. Часто требуется выполнить дополнительную обработку накопленных итогов, например, вычисление средней цены, как частное от деления накопленной суммарной стоимости на суммарное количество. Для реализации такой возможности в MongoDB предусмотрена возможность использования функции финальных вычислений. Например, функция

```
function finalize(key, reducedValue) {
	return {count: reducedValue.count,
		quantity: reducedValue.quantity,
		cost: reducedValue.cost,
		avg_Цена: reducedValue.cost / reducedValue.quantity};
	}
```

к накопленным показателям добавляет вычисление средней цены. Вызов

```
db.Договоры.mapReduce(map, reduce, {finalize: finalize, out:"Goods"});
```

вернет набор

```
{
	"1" : {
		"count" : 50,
		"quantity" : 2801,
		"cost" : 970264,
		"avg_Цена" : 346.399143163156
	},
	"2" : {
		"count" : 55,
		"quantity" : 1840,
		"cost" : 71536,
		"avg_Цена" : 38.87826086956522
	},
	…
}
```

Заключение

Любая информационная система выполняет хранение и обработку данных. Использование определенных структур данных является фундаментом, на котором строятся все основные технологии. Базы данных предоставляют удобный, эффективный, надежный инструмент для решения задач хранения и обработки данных.

При всей универсальности предлагаемых структур каждый тип БД имеет специфику применения. Реляционные БД лучше всего подходят для работы с табличными данными, документационные – для массивов документов, графовые – для описания сложных взаимосвязей и зависимостей, иерархические – для использования древовидных структур. Ясно, что список структур будет расти и дальше, предлагая новые структуры для решения новых классов задач. Очевидно и другое: нет смысла отказываться от известных решений, доказавших свою полезность на практике. Показательной в этом отношении является разработка и применение SQL-подобных средств выбора данных для NoSQL-структур.

Другой тенденцией развития технологии баз данных является применение инструментов распределенного хранения и распределенной обработки данных. Накопление огромных массивов информации во Всемирной паутине просто не может быть выполнено в рамках одного даже очень большого дата-центра. С другой стороны, распределенные хранение и обработка могут оказаться более эффективными благодаря параллельному выполнению.

В целом базы данных на современном этапе развития информационных технологий выступают основным способом структурирования и хранения данных. Знания и умения правильного использования баз данных – необходимая часть профессиональной подготовки ИТ-специалиста.

Список использованной и рекомендуемой литературы

1. Бэнкер К. MongoDB в действии / Кайл Бэнкер. – URL: https://cafe-aristokrat.nethouse.ru/static/doc/0000/0000/0165/165988.c2f3acpbax.pdf.

2. Дейт К.Дж. Введение в системы баз данных / К.Дж. Дейт. – Изд. 6-е. – Москва : Вильямс, 2000. – 846 с.

3. Маклаков С.В. BPWin и ERWin. CASE-средства разработки информационных систем / С.В. Маклаков. – Москва : Диалог-МИФИ, 1999. – 256 с.

4. Марков А.С. Базы данных. Введение в теорию и методологию : учебник / А.С. Марков, К.Ю. Лисовский. – Москва : Финансы и статистика, 2004. – 512 с.: илл.

5. Отраслевой стандарт «Информационные технологии в высшей школе. Термины и определения. ОСТ ВШ 01.002-95». – Москва, 1995. – 24 с.

6. Полубояров В.В. Использование MS SQL Server Analysis Services 2008 для построения хранилищ данных / В.В. Полубояров. – 2-е изд. – Москва : ИНТУИТ, 2016. – 663 с. – URL: http://www.iprbookshop.ru/73682.html.

7. Туманов В.Е. Проектирование хранилищ данных для систем деловой осведомленности (Business Intelligence Systems) / В.Е. Туманов. – Москва : ИНТУИТ, 2016. – 937 с. – URL: http://www.iprbookshop.ru/62825.html.

8. Martin J. Information Engineering: a trilogy. 3 vol. / J. Martin. – Englewood Cliff : Prentice-Hall, 1989–1990.

9. Seguin K. Little MongoDB Book / Karl Seguin. – URL: http://jsman.ru/mongo-book/index.html.

Учебное издание

Братищенко Владимир Владимирович

Реляционные и документационные базы данных

Учебное пособие

Издается в авторской редакции

Компьютерная верстка В.В. Братищенко

ИД № 06318 от 26.11.01.
Подписано в пользование 22.06.20.

Научное издательство Байкальского государственного университета.
664003, г. Иркутск, ул. Ленина, 11.
http://bgu.ru.