

В. Ю. Иномистов

Базы данных
Учебное пособие

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РФ
ГОСУДАРСТВЕННОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ
ВЯТСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
Факультет прикладной математики и телекоммуникаций
Кафедра «Прикладная математика и информатика»

В. Ю. Иномистов
Базы данных.

Рекомендовано Учёным советом
Вятского государственного
университета в качестве учебного
пособия

Киров 2010

Печатается по решению редакционно-издательского совета Вятского государственного университета

УДК 004.65

И67

Иномистов В. Ю. Базы данных./В. Ю. Иномистов. – Киров: Издательство ВятГУ, 2010. – 108 с.

Учебное пособие «Базы данных» для студентов дневного отделения специальности 010500 содержит сведения о проектировании и реализации баз данных Microsoft SQL Server 2008. В пособии рассмотрены вопросы создания объектов баз данных, управления и обеспечения целостности данных, способы оптимизации баз данных, сведения о языке Transact-SQL. Кроме теоретических сведений в пособии представлены задания для лабораторных работ по курсу «Базы данных и экспертные системы» и темы курсовых работ.

Подписано в печать

Усл.печ.л. 6,8

Бумага офсетная

Печать

Заказ №

Тираж

Бесплатно.

Текст напечатан с оригинал-макета, представленного автором

610000, г. Киров, ул. Московская, 36.

Оформление обложки, изготовление – ПРИП ВятГУ

© В. Ю. Иномистов, 2010

© Вятский государственный университет, 2010

Содержание

Введение	4
1. Эволюция систем баз данных	4
1.1. История развития баз данных	4
1.2. История развития SQL	8
1.3. История Microsoft SQL Server	9
2. Задания к лабораторным работам.....	11
2.1. Программные и аппаратные средства	11
2.2 Лабораторная работа № 1. Создание приложения для работы с БД.....	12
2.3. Лабораторная работа № 2. Создание и модификация БД	24
2.4. Лабораторная работа № 3. Создание таблиц	30
2.5. Лабораторная работа № 4. Выполнение простых запросов.....	46
2.6. Лабораторная работа № 5. Выполнение многотабличных запросов	55
2.7. Лабораторная работа № 6. Дополнительные возможности MS SQL.....	57
2.8. Лабораторная работа № 7. Создание триггеров и хранимых процедур	79
2.9. Лабораторная работа № 8. Создание отчётов с использованием Reporting Services.....	89
3. Курсовые работы	105
3.1. Темы курсовых работ.....	105
3.2. Требования к курсовым работам	105
Список использованной литературы.....	107

Введение

Предлагаемое учебное пособие содержит начальные теоретические сведения по курсу «Базы данных и экспертные системы», читаемому студентам специальности 010500 «Прикладная математика и информатика», и задания к лабораторным работам. Кроме этого в пособие включены примерные темы курсовых работ, требования по их выполнению и оформлению.

1. Эволюция систем баз данных

1.1. История развития баз данных

История баз данных (БД – database) началась с полёта человека на Луну. Компания Rockwell заключила с правительством США контракт на участие в проекте Apollo. Построение космического корабля включает в себя сборку нескольких миллионов деталей, поэтому была создана система управления файлами, отслеживавшая информацию о каждой детали. Однако в ходе последующей проверки обнаружилась огромная избыточность. Выяснилось, что почти все данные повторяются в двух и более файлах.

Столкнувшись с задачей координации заказов на миллионы деталей, компания Rockwell в сотрудничестве с IBM в 1968 г. разработала автоматизированную систему заказов. Названная IMS (Information Management System — система управления информацией), она заложила основу концепции системы управления базами данных (СУБД – Database Management Systems - DBMS).

Первая промышленная СУБД – IMS – использовала иерархическую модель данных. Иерархическая модель представляет собой неоднородное дерево (группу деревьев), каждый узел которого обозначает некоторую сущность. Узел может иметь только одного родителя и нуль или более порождённых узлов. Каждая связь «родитель-потомок» означает наличие отношения 1:М между соответствующими сущностями (рисунок 1.1).

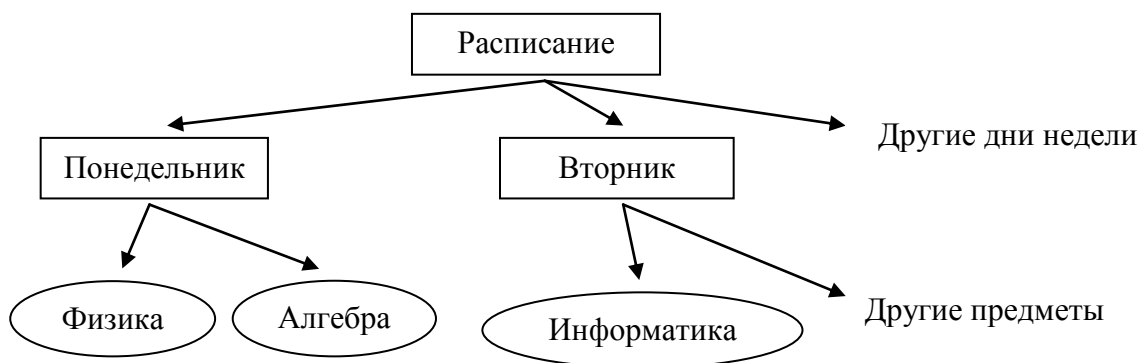


Рисунок 1.1. Пример иерархической модели

Примерно за шесть лет до выпуска IMS была реализована система IDS (Integrated Data Store) фирмы General Electric. Работу над IDS возглавлял Чарльз Бачман (Charles Bachmann). В отличие от продукта IBM в данной системе была реализована сетевая модель данных, лишённая некоторых недостатков иерар-

хической модели. Сетевая модель — произвольно ориентированный граф, возможно с петлями, узлы которого обозначают типы объектов, а дуги — связи между ними.

Примерно к этому же времени относятся первые попытки разработки стандартов БД. На конференции 1965 г. организации CODASYL (Conference on Data Systems Languages) была сформирована группа List Processing Task Force, переименованная в 1967 г. в группу DBTG (Data Base Task Group). В компетенцию группы DBTG входило определение спецификаций среды, которая допускала бы разработку БД и управление данными. Предварительный отчёт был опубликован в 1969 г., окончательный — в 1971 г.

Предложения группы DBTG содержали 3 компонента:

- сетевая схема — это логическая организация всей БД в целом (с точки зрения АБД — администратора БД), которая включает определение имени БД, типа каждой записи и компонентов записей каждого типа;
- подсеть — это часть БД, как она видится пользователям или приложениям;
- язык управления данными — инструмент для определения характеристик и структуры данных, а также для управления ими.

Группа DBTG предложила стандартизировать 3 языка:

- язык определения данных для схемы (Data Definition Language — DDL), который позволяет АБД её описать;
- язык определения данных для подсхемы (DDL), который позволяет определять в приложениях те части БД, доступ к которым будет необходим;
- язык манипулирования данными (Data Manipulation Language — DML), предназначенный для управления данными.

Отчёт не был официально утвержден Национальным институтом стандартизации США (ANSI), но, тем не менее, стал основой для ряда успешных коммерческих разработок. Наиболее успешным был продукт корпорации Cullinane под названием IDMS.

CODASYL-системы (DBTG-системы) и системы на основе иерархических подходов представляют собой СУБД первого поколения.

Общие недостатки:

- даже для выполнения простых запросов с использованием переходов и доступом к определённым записям необходимо создавать достаточно сложные программы;
- независимость от данных существует лишь в минимальной степени;
- отсутствие общепризнанных теоретических основ.

В 1970 году Эдгар Франк Кодд (Edgar F. Codd) — исследовательская лаборатория корпорации IBM — опубликовал статью «Реляционная модель данных для больших совместно используемых банков данных» (Codd E.F. A relational model for large shared data banks, Comm. ACM, 13:6 (1970), p. 377-387), которая считается поворотным пунктом в истории развития систем БД. Реляционная модель позволяла устранить недостатки прежних моделей. Для проверки практичности реляционной БД был создан исследовательский проект System R (исследова-

тельская лаборатория IBM, Сан-Хосе, штат Калифорния, конец 1970-х годов). В результате работы над проектом был создан язык SEQUEL (Structured English QUEry Language – «структурированный английский язык запросов»). По юридическим соображениям («SEQUEL» – торговая марка британской авиастроительной группы компаний Hawker Siddeley) позже он был переименован в SQL (Structured Query Language – «структурированный язык запросов»).

Реляционные системы образовали СУБД второго поколения. Первыми коммерческими реляционными СУБД стали появившиеся в 1979 году Oracle V2 для машин VAX от компании Relational Software Inc. (впоследствии ставшей компанией Oracle) и System/38 от IBM, основанная на System R. При этом продукт Oracle вышел на пару месяцев раньше СУБД от IBM.

В настоящее время наиболее распространёнными являются СУБД трёх производителей. Это Oracle (компания Oracle), DB2 (IBM) и SQL Server (Microsoft).

В реляционной модели БД представляет собой централизованное хранилище таблиц, обеспечивающее безопасный одновременный доступ к информации со стороны многих пользователей. В строках таблиц часть полей содержит данные, относящиеся непосредственно к записи, а часть — ссылки на записи других таблиц. Таким образом, связи между записями являются неотъемлемым свойством реляционной модели.

Каждая запись таблицы имеет одинаковую структуру. Такие таблицы легко изображать в графическом виде (рисунок 1.2).

Наименование ВУЗа	Краткое наименование	Город	Код города
Нижегородский государственный университет	ННГУ	Нижний Новгород	НН
Санкт-Петербургский государственный университет	СПбГУ	Санкт-Петербург	СПб
Вятский государственный университет	ВятГУ	Киров	Киров

Рисунок 1.2. Пример реляционной модели

В реляционной модели достигается информационная и структурная независимость. Записи не связаны между собой настолько, чтобы изменение одной из них затронуло остальные, а изменение структуры БД не обязательно приводит к перекомпиляции работающих с ней приложений.

В реляционных СУБД применяется язык SQL, позволяющий формулировать произвольные, нерегламентированные запросы.

До 1990-х годов реляционные СУБД занимали главенствующее положение. Однако неудобство реляционной модели для хранения мультимедийных данных, разработки сложных информационных прикладных систем привело к появлению СУБД третьего поколения, к которым относятся Объектно-ориентированные СУБД (ООСУБД – Object-Oriented DBMS – OODBMS) и

Объектно-реляционные СУБД (ОРСУБД – Object-Relational DBMS – ORDBMS).

Преимуществом ООБД является упрощённый код. Приложения получают возможность интерпретировать данные в контексте того языка программирования, на котором они написаны. В ООБД методы манипулирования данными всегда остаются одинаковыми независимо от того, находятся данные на диске или в памяти. Данные в ООБД способны принять вид любой структуры, которую можно выразить на используемом языке программирования. Отношения между сущностями так же могут быть произвольно сложными. ООБД управляет кэш-буфером объектов, перемещая объекты между буфером и дисковым хранилищем по мере необходимости.

С помощью ООБД решаются две проблемы. Во-первых, сложные информационные структуры выражаются в них лучше, чем в реляционных БД, а во-вторых, устраняется необходимость транслировать данные из того формата, который поддерживается в СУБД.

ООСУБД выполняют много дополнительных функций. Это окупается сполна, если отношения между данными очень сложны. В таком случае производительность ООСУБД оказывается выше, чем у реляционных СУБД. Если же это не так, дополнительные функции оказываются избыточными.

В объектной модели данных поддерживаются нерегламентированные запросы, но языком их составления не обязательно является SQL. Логическое представление данных может не соответствовать реляционной модели, что делает применение языка SQL бессмысленным. Зачастую удобнее обрабатывать объекты в памяти, выполняя соответствующие виды поиска.

Большим недостатком ООБД является их тесная связь с применяемым языком программирования. К данным, хранящимся в реляционной СУБД, могут обращаться любые приложения, тогда как, к примеру, Java-объект, помещённый в ООБД, будет представлять интерес лишь для приложений, написанных на Java.

ОРСУБД объединяют в себе черты реляционной и объектной моделей. Их возникновение объясняется тем, что реляционные БД хорошо работают со встроенными типами данных и гораздо хуже — с пользовательскими, нестандартными. Когда появляется новый важный тип данных, приходится либо включать его поддержку в СУБД, либо заставлять программиста самостоятельно управлять данными в приложении. Не всякую информацию имеет смысл интерпретировать в виде цепочек символов или цифр.

Пример. Музыкальная база данных. Песню, закодированную в виде аудио-файла, можно поместить в текстовое поле большого размера, но как в таком случае будет осуществляться текстовый поиск?

Перестройка СУБД с целью включения в неё поддержки нового типа данных — не лучший выход из положения. Вместо этого ОРСУБД позволяет загружать код, предназначенный для обработки «нетипичных» данных. Таким образом, БД сохраняет свою табличную структуру, но способ обработки некоторых полей таблиц определяется извне, т.е. программистом.

1.2. История развития SQL

Появление языка SQL связано с экспериментальной СУБД System R корпорации IBM. В 1974 году Дональд Чемберлен (Donald D. Chamberlin) опубликовал определение SEQUEL. Основная идея при разработке этого языка заключалась в его максимальной близости семантике английского языка, что позволяло (в теории) сделать его инструментом конечного пользователя, не обладающего навыками программирования. Первоначальная версия языка вполне удовлетворяла заявленной цели, что позволило ей победить в конкурентной борьбе QUEL (СУБД Ingres – прародитель СУБД PostgreSQL), который являлся более структурированным, чем SQL, но менее близким английскому языку.

Корни SQL – язык SQUARE (Specifying Queries as Rational Expressions), разработанный как исследовательский инструмент для реализации реляционной алгебры посредством фраз, составленных на английском языке.

В 1982 году ANSI начал работу над языком RDL (Relation Database Language). В 1983 году к нему присоединилась ISO. Результатом их совместной работы стал исходный вариант стандарта SQL (от названия RDL в 1984 году отказались, а проект языка переработали с целью приближения к уже существующим реализациям SQL), появившийся в 1987 году.

Этот вариант SQL вызвал волну критики. В частности указывалось на то, что в стандарте опущены важнейшие функции (средства обеспечения ссылочной целостности, некоторые реляционные операторы), имеется чрезмерная избыточность языка (один и тот же результат можно получить с помощью различных запросов). Критика была признана справедливой, и некоторые изменения были внесены в стандарт ещё до его публикации, но вместе с тем была отмечена необходимость его скорейшей публикации.

В 1989 году ISO опубликовала дополнение к стандарту, в котором определялись функции поддержки целостности данных.

В 1992 году была выпущена существенно пересмотренная версия стандарта (SQL2 или SQL-92). Многие функции, впервые определённые в этом стандарте, были уже частично или полностью реализованы в коммерческих продуктах.

В 1999 году появился стандарт SQL:1999 (SQL3), содержащий дополнительные средства поддержки объектно-ориентированных функций управления данными.

В 2003 году вышел стандарт SQL:2003. В нём, в частности, введены расширения для работы с XML-данными, оконные функции, генераторы последовательностей и основанные на них типы данных.

В 2006 и 2008 годах увидели свет версии стандарта SQL:2006 и SQL:2008.

Несмотря на наличие стандарта, разработчики СУБД стремятся увеличить возможности языка SQL, добавляя новые функции, которые носят название «расширение». Каждая реализация SQL называется «диалектом». Несмотря на это, в целом язык SQL можно признать независимым от конкретной СУБД. Вернее, большинство SQL-запросов может быть легко адаптировано для работы в другой СУБД.

Основные достоинства языка SQL могут быть сведены к следующим пунктам:

- независимость (относительная) от конкретной СУБД;
- наличие стандартов;
- декларативность, т.е. необходимо указывать, какая информация должна быть получена, а не как её получить.

К недостаткам SQL следует отнести следующее:

- несоответствие реляционной модели данных (!):
 - повторяющиеся строки,
 - неопределённые значения (NULL),
 - явное указание порядка колонок слева направо,
 - колонки без имени и дублирующиеся имена колонок,
 - использование указателей,
 - высокая избыточность (так и не преодолённая со времён стандарта SQL-87);
- сложность — задуманный как простой и доступный язык для конечного пользователя, он превратился в сложный инструмент, требующий серьёзной подготовки;
- отступления от стандартов — наличие диалектов, не вполне отвечающих текущему или предшествующим стандартам;
- сложность работы с иерархическими структурами.

1.3. История Microsoft SQL Server

В 1986 году Microsoft и Sybase выпустили совместную версию продукта — SQL Server 1.0 и адаптировали её для операционной системы OS/2 при поддержке компании Ashton-Tate, которая в то время была лидером на рынке СУБД для персональных компьютеров. Выпущенный в 1989 году продукт не получил должного признания из-за проблем, связанных с продвижением OS/2.

В 1990 году Sybase и Microsoft прервали соглашение с Ashton-Tate и выпустили версию SQL Server 1.1 для новой операционной системы Windows 3.0. Microsoft отвечала за клиентские утилиты, программные интерфейсы и средства управления, а Sybase — за разработку ядра БД.

В 1992 году началась разработка новой версии продукта — SQL Server on Windows NT, который был выпущен в 1993 году одновременно с серверной операционной системой — Microsoft Windows NT. Тесная интеграция с Windows NT обеспечила продукту высокую производительность, управляемость и впервые у Microsoft появилась СУБД, которая могла конкурировать с аналогичными продуктами на платформе UNIX.

В 1994 году Microsoft и Sybase прервали совместное пятилетнее соглашение, и бывшие партнёры занялись самостоятельным развитием своих, теперь уже конкурирующих, продуктов.

В 1995 и 1996 годах увидели свет версии SQL Server 6.0 и 6.5, но некоторые проблемы с производительностью и управляемостью не позволили этим про-

дуктам завоевать существенную долю рынка корпоративных СУБД. Было принято решение приостановить развитие текущей версии платформы и начать создание продукта «с нуля». Примерно в то же время компания DEC продала свою СУБД компании Oracle и Microsoft удалось заполучить ведущих специалистов компании DEC — Джима Грея (Jim Gray), Дэйва Ломета (Dave Lomet) и Фила Бернштейна (Phil Bernstein).

Команде разработчиков была поставлена задача — создать новое ядро БД с поддержкой масштабируемости, новый процессор обработки запросов, систему самонастройки, самоуправления, а также реализовать поддержку OLAP (online analytical processing – аналитическая обработка в реальном времени) и ETL (Extract, Transform, Load – извлечение, преобразование, загрузка) с привлечением специалистов из компании Panorama. Разработка новой СУБД заняла около трёх лет и в 1998 году был выпущен продукт под названием SQL Server 7.0 — Microsoft начала завоевывать не только рынок реляционных СУБД, но и такие новые рынки, как business intelligence и data warehousing.

Параллельно велась работа над SQL Server 2000, который включал в себя поддержку XML, индексированные представления, распределённые разделы на основе представлений, а также более чем 20%-ное увеличение производительности для практически всех ключевых компонентов продукта. В 2000 году Microsoft стала полноправным лидером на рынке СУБД для платформы Windows.

Дальнейшее развитие продукта — в версиях SQL Server 2005 и SQL Server 2008 — добавило увеличение производительности, управляемости, расширенную поддержку различных типов данных, интегрированные системы создания отчётов, трансформации данных, расширенные функции анализа и т. п.

При этом версию MS SQL Server 2005 принято относить к революционным, так как в ней впервые появились такие новшества (для MS SQL Server), как секционированные таблицы, схемы, исполнение хранимых процедур от имени, Service Broker, DDL-триггеры и многое другое. Всё это не могло не сказаться на сроках выхода новой версии. Можно заметить, что выход нового MS SQL Server по времени должен был совпадать с появлением новой версии MS Windows Server и появлением нового стандарта SQL. Это и неудивительно, работа SQL Server довольно сильно связана с операционной системой, и их тесная интеграция благотворно влияет на производительность и надёжность СУБД. Выход SQL Server почти сразу после появления нового стандарта позволяет (если возможно) заявить о хорошем соответствии новой СУБД новому стандарту. Впрочем, Microsoft далеко не всегда делится с конечным пользователем информацией о поддержке своими продуктами стандартов.

В соответствии с наметившейся к 2000 году традицией, следовало ожидать выхода очередной версии MS SQL Server в 2003 году одновременно с появлением MS Windows Server 2003 и MS Visual Studio 2003, но этого не произошло. Основная причина того, что вместо SQL Server 2003 появился SQL Server 2005 (в 2005 году, как и следует из его названия), проста и понятна: в Microsoft просто не успели довести до работоспособного состояния новую версию СУБД.

Так увидела свет достаточно нетрадиционная по времени появления версия MS SQL Server 2005.

Повторять ещё раз такое с новой версией СУБД Microsoft не стала, и MS SQL Server 2008 появился в один год с новой версией MS Windows Server 2008, хотя и проходил презентацию почти на полгода раньше своего официального выхода.

Новую СУБД от Microsoft относят к эволюционному развитию MS SQL Server 2005. Действительно, каких-то революционных новинок достаточно мало (чуть ли не единственной серьёзной новинкой стала балансировка нагрузки – Resource Governor), большинство изменений либо касаются отказа от некоторых устаревших элементов (тип данных text), либо являются развитием идей и методов, заложенных в MS SQL Server 2005.

Как бы то ни было, выход новой версии СУБД от Microsoft сделал её ещё чуть более мощной, чуть более продвинутой, чуть более близкой к СУБД третьего поколения. Последнее, на самом деле, даже не очень-то и чуть-чуть. В MS SQL Server 2008 появилась возможность хранить и обрабатывать мультимедийные данные произвольного типа и объёма.

2. Задания к лабораторным работам

2.1. Программные и аппаратные средства

Лабораторные работы по курсу «Базы данных и экспертные системы» проводятся в аудитории, оснащённой вычислительной техникой, конфигурация которой позволяет организовать запуск виртуальной машины с помощью программы Microsoft Virtual PC 2007. Виртуальная машина представляет из себя операционную систему MS Windows XP Professional с установленным MS SQL Server 2008 Developer Edition. Такая конфигурация является минимальной. Для обеспечения приемлемого быстродействия достаточно удовлетворение следующих минимальных системных требований:

- процессор не хуже Intel Pentium III 600 МГц;
- свободное дисковое пространство не менее 1,6 Гбайта;
- оперативная память не менее 1 Гбайта.

Наиболее критичным является последнее требование, так как Microsoft Virtual PC 2007 использует для загрузки виртуальной машины только оперативную память компьютера. Именно поэтому, например, использование в качестве операционной системы для виртуальной машины MS Windows Server 2008 представляется малореальным, так как для комфортной работы требуется больший объём памяти, нежели имеется на компьютерах в дисплейных классах.

Использование виртуальных машин вместо установки необходимого программного обеспечения на основную операционную систему придаёт большую гибкость при проведении лабораторных работ:

- под виртуальной машиной пользователь (студент) может иметь права администратора компьютера;

- любые неудачные действия студента не могут привести к неработоспособности основной системы;
- возможно конфигурирование СУБД под задачи конкретной лабораторной работы;
- отсутствие необходимости приведения системы в первоначальное состояние после окончания лабораторной работы;
- отсутствие каких-либо конфликтов с другим системным и учебным программным обеспечением.

Выбор в качестве конкретной СУБД MS SQL Server 2008 не является случайным и базируется на следующих соображениях:

- СУБД от Microsoft являются достаточно распространёнными;
- данная СУБД обладает всем необходимым для изучения возможностей современных мощных СУБД;
- данная СУБД относится к лицензионным (для ВУЗа) программным продуктам;
- по СУБД от Microsoft имеется большой объём учебного материала (книги, электронные ресурсы, в том числе и на сайте Microsoft);
- знания теоретических основ БД и навыки работы с MS SQL Server 2008 позволяют легко адаптироваться практически к любой другой СУБД, будь то Oracle, DB2, PostgreSQL или MySQL.

Для выполнения лабораторной работы студенты получают настроенную необходимым образом виртуальную машину. Получение навыков установки, конфигурирования и администрирования СУБД в данном курсе не предусматривается.

Практическая часть курса состоит из восьми лабораторных работ продолжительностью четыре часа каждая. Часть лабораторных работ содержит индивидуальные задания, часть направлена на получение стандартных навыков работы с БД и содержит одинаковые для всех задания.

Выполнение индивидуальных заданий подразумевает создание отчёта по лабораторной работе, в который включаются необходимый теоретический материал и текст созданных в ходе работы запросов, а также, при необходимости, структуры таблиц с примером их заполнения, связи между таблицами.

2.2 Лабораторная работа № 1. Создание приложения для работы с БД

Цель работы: знакомство со способом создания приложения в среде Visual Studio 2008 для работы с БД.

Содержание работы

1. Создание приложения.

Для создания приложения, работающего с БД, будет использоваться MS Visual Studio 2008. Проект создаётся на языке Visual C#. Для начала работы выбирается шаблон «Приложение Windows Forms» (рисунок 2.1).

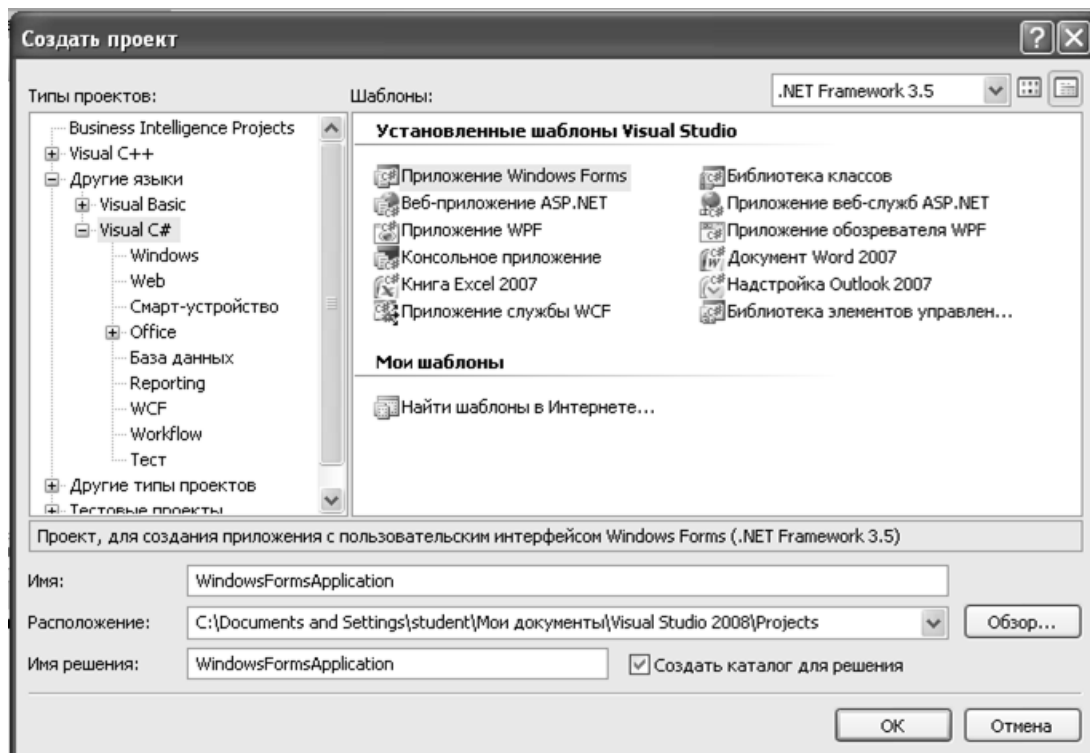


Рисунок 2.1. Окно выбора шаблона.

На следующем шаге подключается источник данных (в данном случае это будет БД). Для этого необходимо выбрать пункт меню **Данные** → **Добавить новый источник данных**. В появившемся окне выбирается источник данных **База данных** (рисунок 2.2).

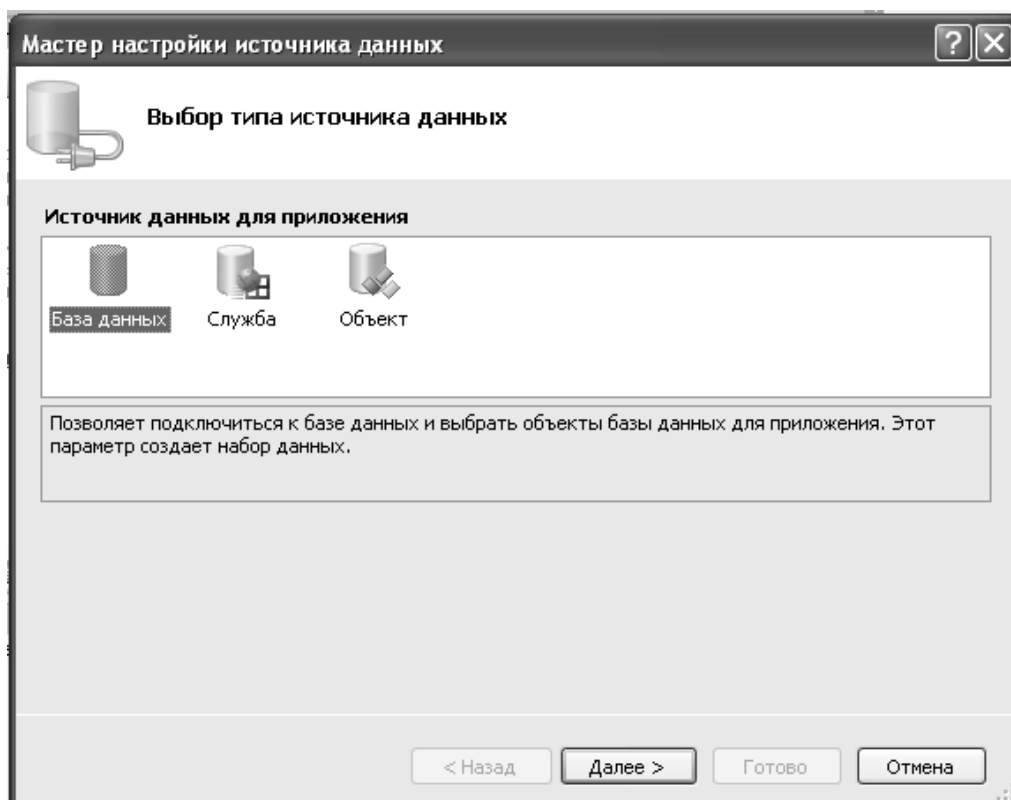


Рисунок 2.2. Окно выбора источника данных.

При переходе к следующим окнам настраивается выбор подключения (рисунки 2.3 – 2.5) и выбор объектов БД (рисунок 2.6).

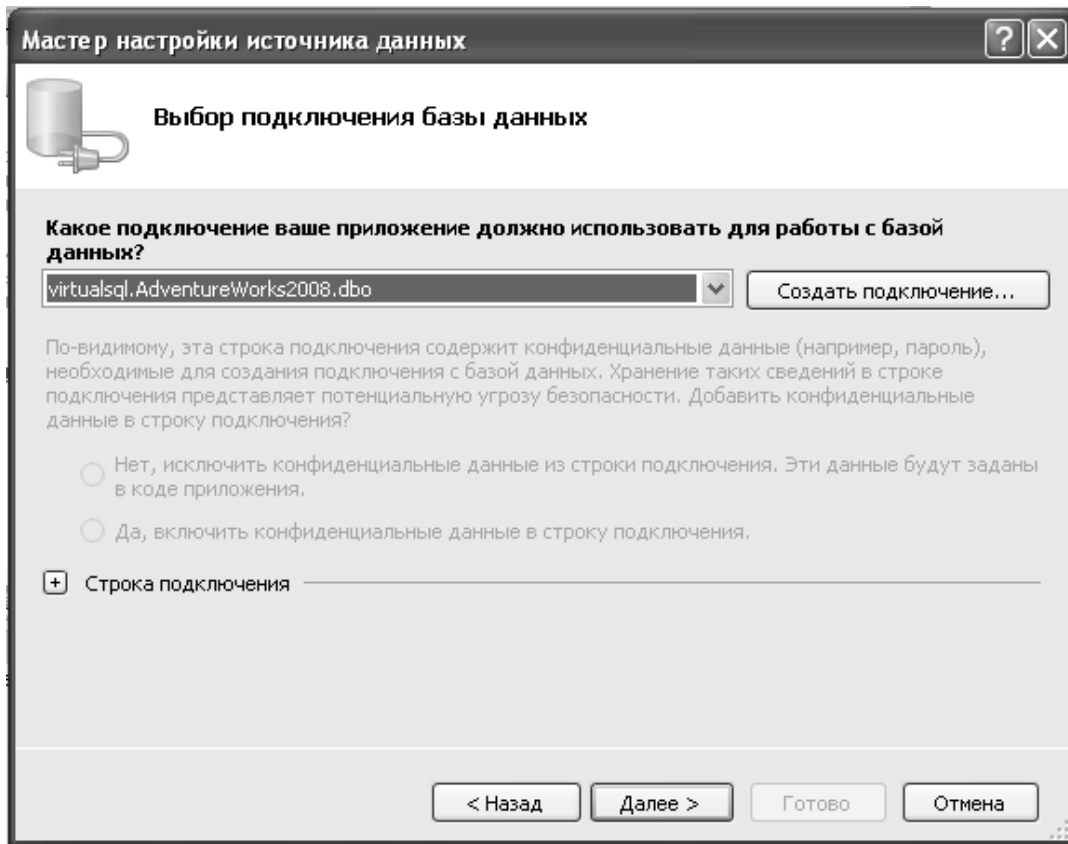


Рисунок 2.3. Выбор подключения БД

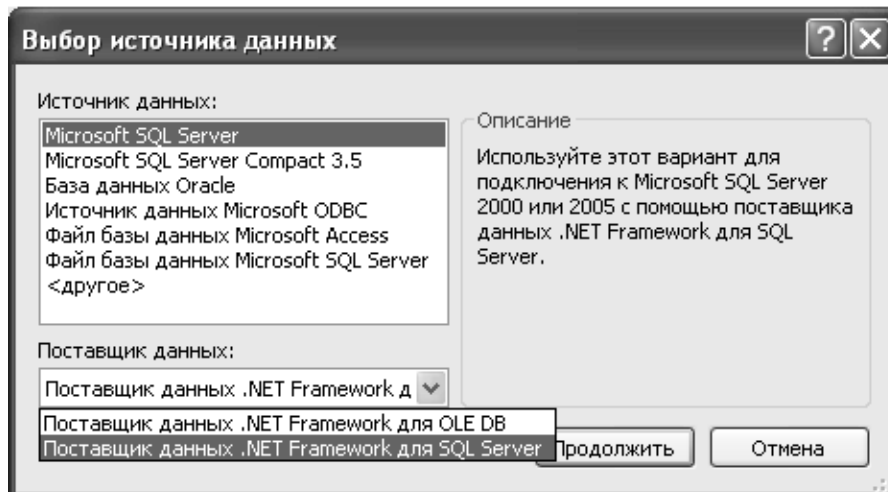


Рисунок 2.4. Добавление нового соединения.

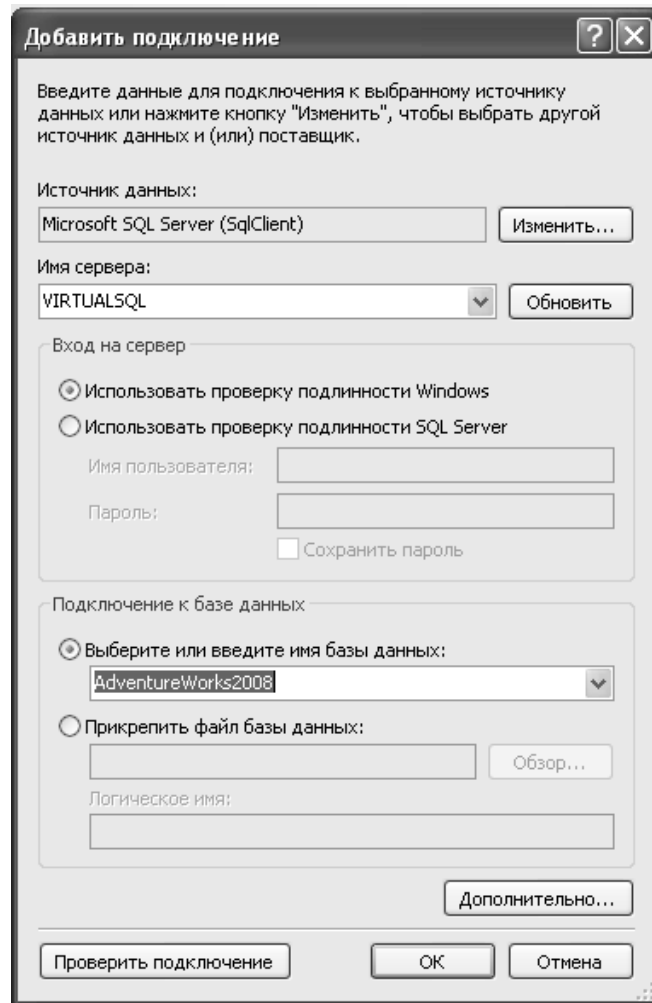


Рисунок 2.5. Добавление подключения.

В мастере настройки источника данных необходимо выбрать те объекты, с которыми будет работать создаваемое приложение. Это могут быть не только таблицы, но и отдельные поля, представления, хранимые процедуры и функции. В данной лабораторной работе будут использоваться таблицы **Person.Person** и **Person.PersonPhone**.

По окончании процесса подключения данных выбранная БД подсоединится к приложению, и в окне **Источники данных** появятся выбранные объекты (рисунок 2.7).

В результате создаётся набор данных (НД), который носит название AdventureWorks2008DataSet. НД задаёт источник данных, запрос, который является командой на получение данных из этого источника, параметры запроса, фильтры и коллекцию полей, представляющую результирующий набор. Через НД происходит подключение приложений к БД. При этом информация из БД загружается в НД, который затем станет обеспечивать данными приложение из локальной кэш-памяти. С таким набором данных можно работать даже тогда, когда приложение отсоединено от БД. НД поддерживает информацию об изменениях таким образом, что обновленные данные могут быть снова отосланы БД, когда приложение подсоединится к ней.

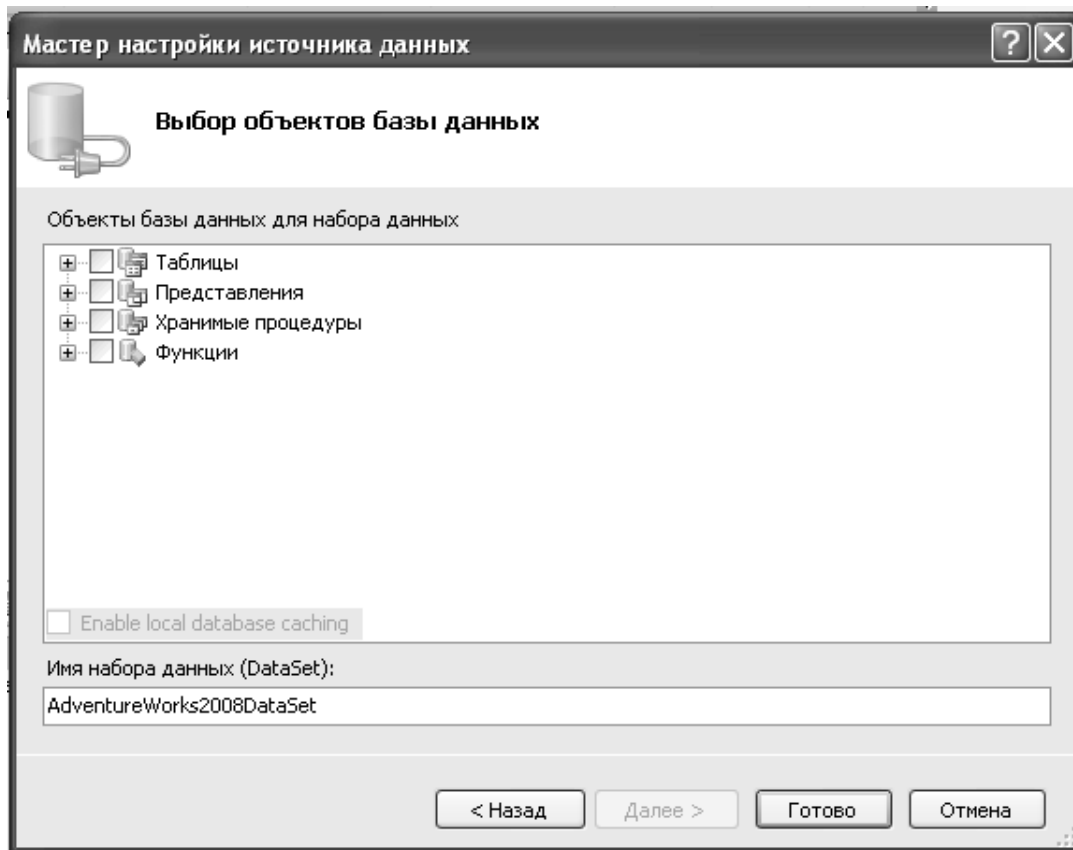
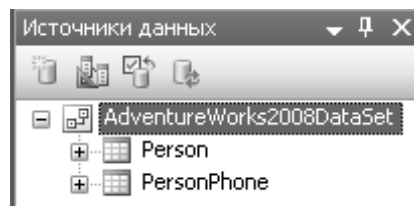


Рисунок 2.6. Выбор объектов БД.

Рисунок 2.7. Окно **Источники данных** с выбранными таблицами.

Контекстное меню окна **Источники данных** содержит пункты, позволяющие добавить новый источник данных, изменить или настроить имеющийся НД. При выборе пункта меню **Изменить набор данных в конструкторе** откроется вкладка, на которой будут представлены те таблицы, которые содержатся в НД. Внизу для каждой таблицы содержится компонент **TableAdapter**. Именно с помощью этого компонента и осуществляется пересылка данных между НД и БД посредством специальных методов заполнения и выборки из таблицы.

При выборе пункта контекстного меню **Свойства** для **TableAdapter** откроется соответствующее окно (рисунок 2.8).

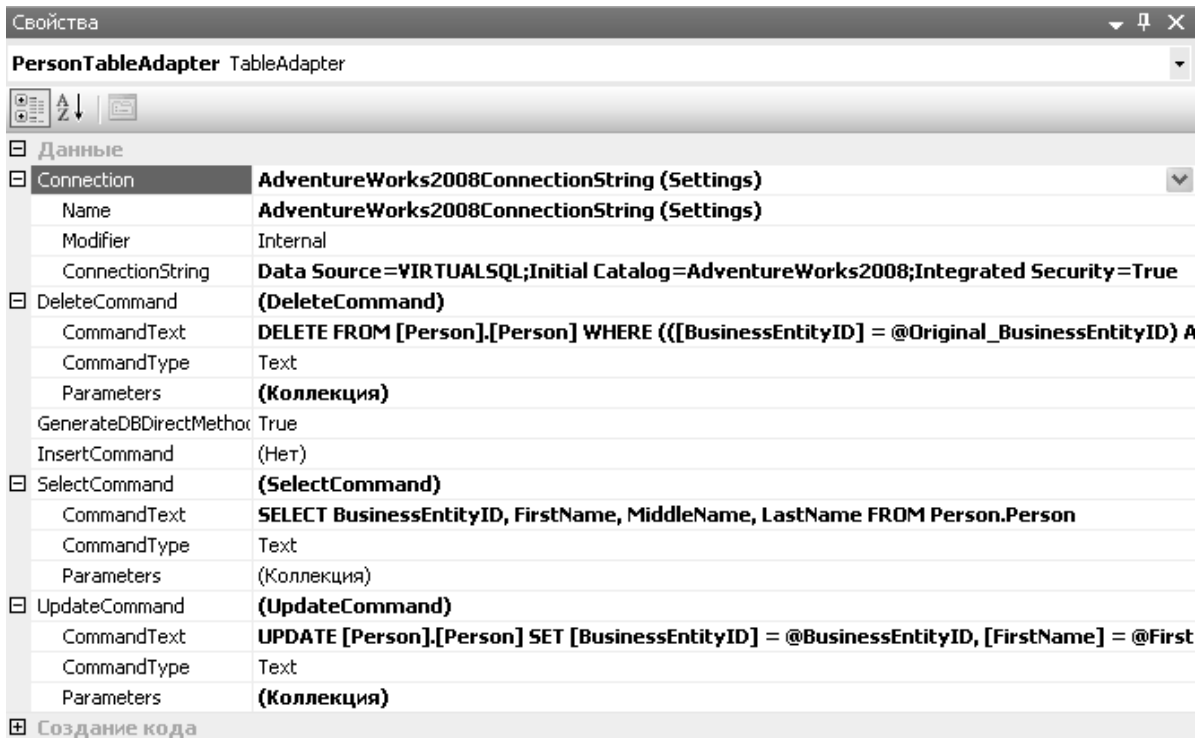


Рисунок 2.8. Свойства компонента TableAdapter.

Можно заметить, что для команд **DeleteCommand**, **InsertCommand**, **SelectCommand**, **UpdateCommand** существуют такие свойства, как **CommandText** – текст команды, **CommandType** – тип команды, **Parameters** – параметры. Тип команды можно выбрать из предложенного списка – **Text**, **Stored Procedure** или **TableDirect**.

Для отображения данных будет использоваться компонент **DataGridView** (выпадающий список при выборе таблицы в окне **Источники данных**) – это так называемая сетка или решётка, являющаяся средством табличного представления данных, помещаемых на форму. На первом этапе рассмотрен вариант отображения одной таблицы – **Person.Person**. Необходимо поместить требуемую таблицу на форму. При этом среда программирования автоматически сформирует дополнительные компоненты, обеспечивающие взаимодействие формы с НД. Это, прежде всего, компонент **DataSet**, компонент **BindingSource**, обеспечивающий связь с соответствующей таблицей БД, компонент **BindingNavigator**, и, наконец, компонент **TableAdapter**, непосредственно исполняющий запросы к каждой таблице (рисунок 2.9).

Замечание. Кроме компонента **DataGridView** с уже определённым набором полей на форме появляется и компонент **BindingNavigator**, содержащий кнопки для управления таблицей.

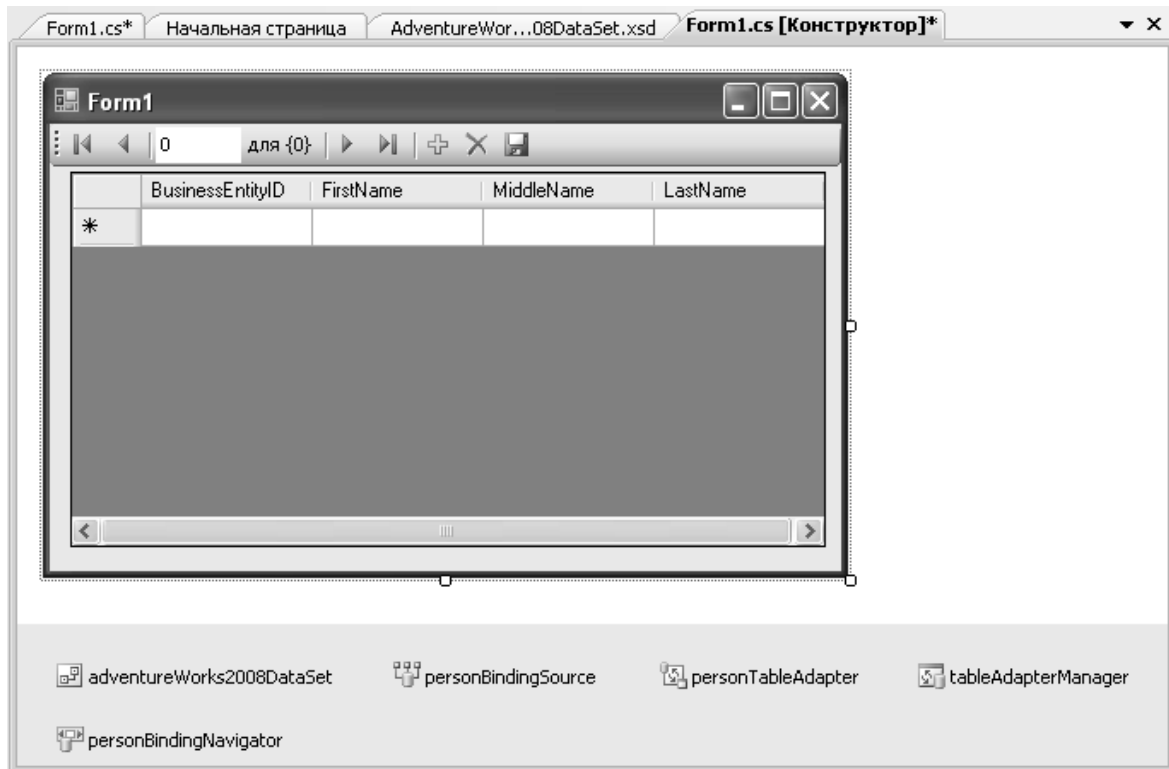


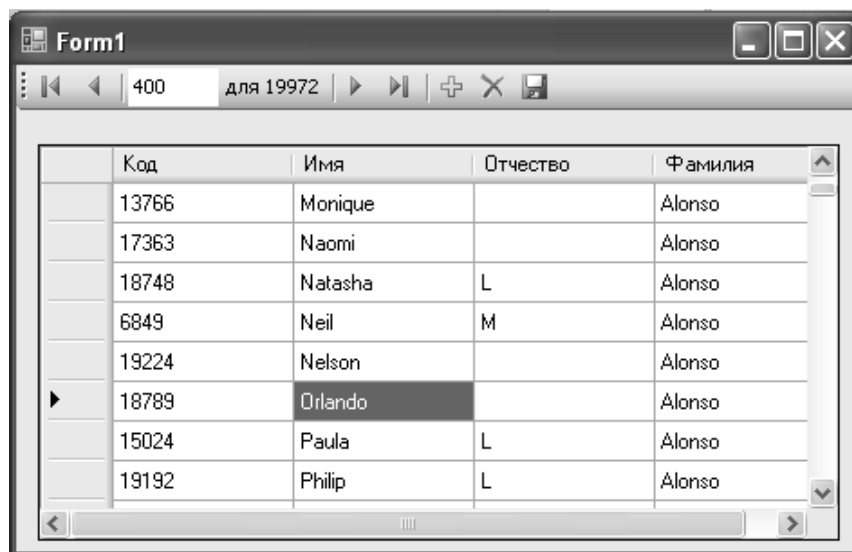
Рисунок 2.9. Форма для отображения данных.

При выборе пункта контекстного меню **Правка столбцов** компонента **DataGridView** появляется возможность настроить список отображаемых столбцов, порядок их отображения, заголовки (свойство **HeaderText**) и некоторые другие свойства, удалить их или сделать просто невидимыми.

Далее рассматривается работа с компонентом **BindingNavigator**. Для перемещения по строкам таблицы служат 4 кнопки со стрелками: **MoveFirstItem**, **MovePreviousItem**, **MoveNextItem**, **MoveLastItem**, в поле **PositionItem** показывается позиция текущей строки. Далее идёт кнопка **AddNewItem**. По нажатию на ней в таблице появляется новая строка, куда нужно вводить данные. Кнопка **DeleteItem** удаляет из таблицы текущую строку. Кроме того, значения в таблице можно редактировать. Для этого нужно установить курсор в нужном поле и изменить значение. Все изменения, которые претерпела таблица – добавление новых строк, редактирование, удаление – сохранились только в НД **DataSet**. Для переноса этих изменений из НД в БД используется кнопка **SaveItem**.

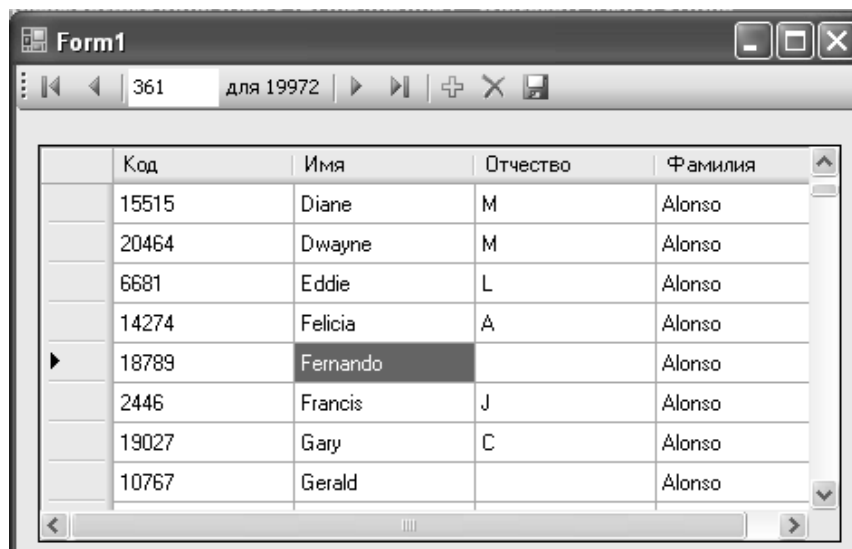
При запуске приложения будут отображены все данные, которые содержатся в таблице. Можно добавлять новые данные, редактировать имеющиеся, удалять ненужные. В среде **Microsoft SQL Server Management Studio** можно убедиться, что данные в БД действительно изменяются (рисунки 2.10 – 2.12).

При внесении изменений в таблицу необходимо помнить о тех ограничениях, которые предусмотрены в БД (например, о том, что некоторые поля должны быть обязательно заполнены). Если при изменении данных из приложения были нарушены условия, накладываемые на соответствующее поле в таблице БД, то будет выдано сообщение о произошедшем исключении (рисунок 2.13).



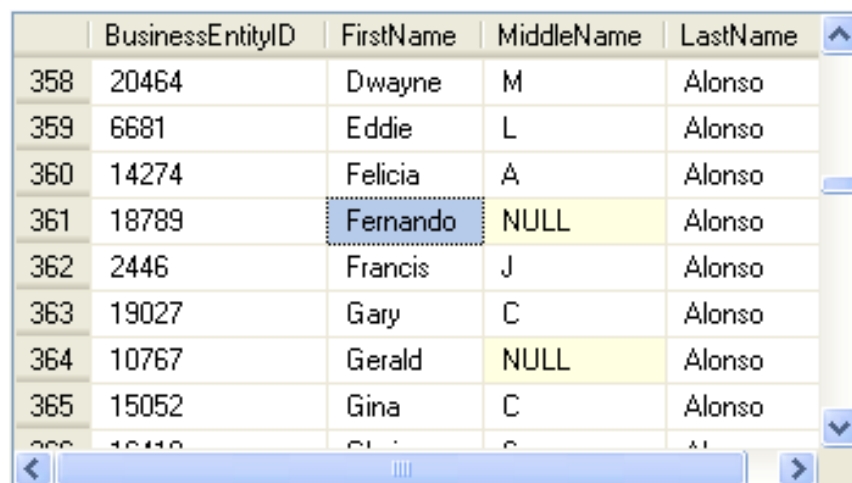
	Код	Имя	Отчество	Фамилия
	13766	Monique		Alonso
	17363	Naomi		Alonso
	18748	Natasha	L	Alonso
	6849	Neil	M	Alonso
	19224	Nelson		Alonso
▶	18789	Orlando		Alonso
	15024	Paula	L	Alonso
	19192	Philip	L	Alonso

Рисунок 2.10. Данные до изменения.



	Код	Имя	Отчество	Фамилия
	15515	Diane	M	Alonso
	20464	Dwayne	M	Alonso
	6681	Eddie	L	Alonso
	14274	Felicia	A	Alonso
▶	18789	Fernando		Alonso
	2446	Francis	J	Alonso
	19027	Gary	C	Alonso
	10767	Gerald		Alonso

Рисунок 2.11. Данные после изменения.



	BusinessEntityID	FirstName	MiddleName	LastName
358	20464	Dwayne	M	Alonso
359	6681	Eddie	L	Alonso
360	14274	Felicia	A	Alonso
361	18789	Fernando	NULL	Alonso
362	2446	Francis	J	Alonso
363	19027	Gary	C	Alonso
364	10767	Gerald	NULL	Alonso
365	15052	Gina	C	Alonso

Рисунок 2.12. Фрагмент таблицы с изменёнными данными.

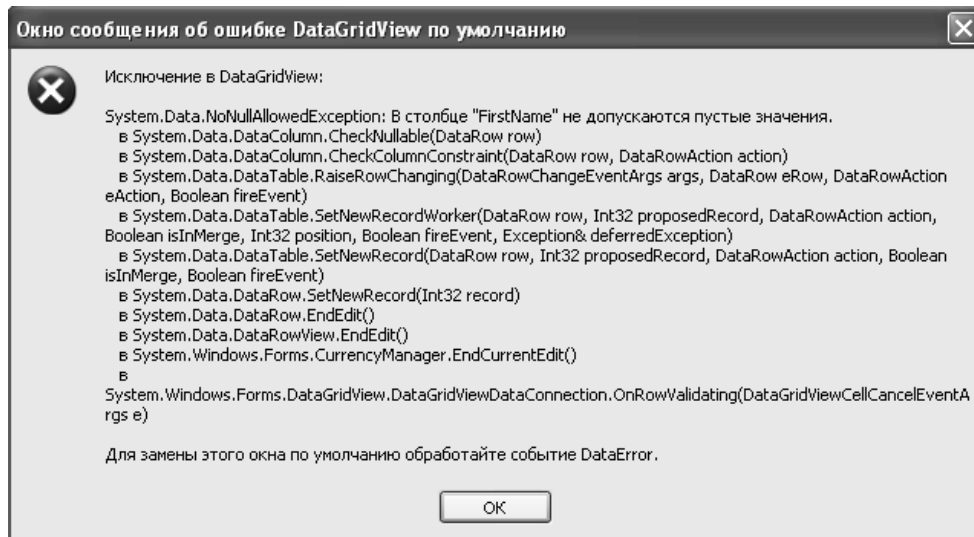


Рисунок 2.13. Окно сообщения об ошибке.

Исключения можно отслеживать и обрабатывать. Например, по умолчанию код обработки события, которое возникает при нажатии на кнопку **SaveItem**, выглядит следующим образом:

```
private void personBindingNavigatorSaveItem_Click(object sender, EventArgs e)
{
    this.Validate();
    this.personBindingSource.EndEdit();
    this.tableAdapterManager.UpdateAll(this.adventureWorks2008DataSet);
}
```

Чтобы обработать исключения, нужно данный код поместить в конструкцию try-catch. Обработкой исключения в данном случае будет вывод сообщения исключения и перезаполнение таблицы. Код будет выглядеть следующим образом:

```
private void personBindingNavigatorSaveItem_Click(object sender, EventArgs e)
{
    try
    {
        /* Здесь содержится код, в котором может возникнуть исключение */
        this.Validate();
        this.personBindingSource.EndEdit();
        this.tableAdapterManager.UpdateAll(this.adventureWorks2008DataSet);
    }
    catch (Exception exp1)
    {
        /* В скобках исключение типа Exception указывает на то, что в этом блоке
        обрабатываются любые исключения */
        /* Сам код обработчика исключения */
        MessageBox.Show(exp1.Message.ToString(), "Исключение",
```

```

        MessageBoxButtons.OK, MessageBoxIcon.Error);    //Вывод сообщения
    исключения
    /* Перезаполнение таблицы, изменения не будут сохранены */
    //Освобождение памяти
    this.personTableAdapter.Dispose();
    //Заполнение
    this.personTableAdapter.Fill(this.adventureWorks2008DataSet.Person);
}
}

```

Выше был рассмотрен случай, когда требовалось отобразить данные из одной таблицы. Часто оказывается, что одна таблица связана внешними ключами с другими. В этом случае может понадобиться отобразить в одном компоненте **DataGridView** данные из нескольких связанных таблиц.

Пусть в БД имеются две таблицы. Одна таблица **Person.Person** содержит следующие поля: **BusinessEntityID** (первичный ключ), **FirstName**, **MiddleName** и **LastName**. Другая таблица **Person.PersonPhone** содержит поля **BusinessEntityID** (внешний ключ) и **PhoneNumber**. Связь между таблицами осуществляется через указанный внешний ключ.

Замечание 2. Выше перечислены не все поля, входящие в указанные таблицы, а только те, которые выбраны для дальнейшей работы.

Обе таблицы уже присутствуют в НД, но в более общем случае они могли бы быть добавлены в существующий НД способом, аналогичным описанному выше.

Для работы будет создана новая форма, на которой будут отображаться данные из обеих таблиц. Первоначальные шаги аналогичны тем, что были описаны для примера с отображением данных из одной таблицы. В **Источнике данных** выбирается таблица **Person**, для неё устанавливается компонент **DataGridView** и таблица помещается на форму (также должен появиться **BindingNavigator**).

Теперь необходимо сделать настройки столбцов. Для этого из контекстного меню в поле **DataGridView** выбирается пункт **Правка столбцов**, после чего появится окно, в котором будут отражены поля таблицы. Для рассматриваемого случая необходимо создать новый столбец. Для вновь созданного столбца свойство **ColumnType** устанавливается в **DataGridViewComboBoxColumn** (рисунок 2.14), после чего в списке появятся такие свойства как **DataPropertyName**, **DataSources**, **DisplayMember**, **ValueMember**, **Items** (рисунок 2.15).

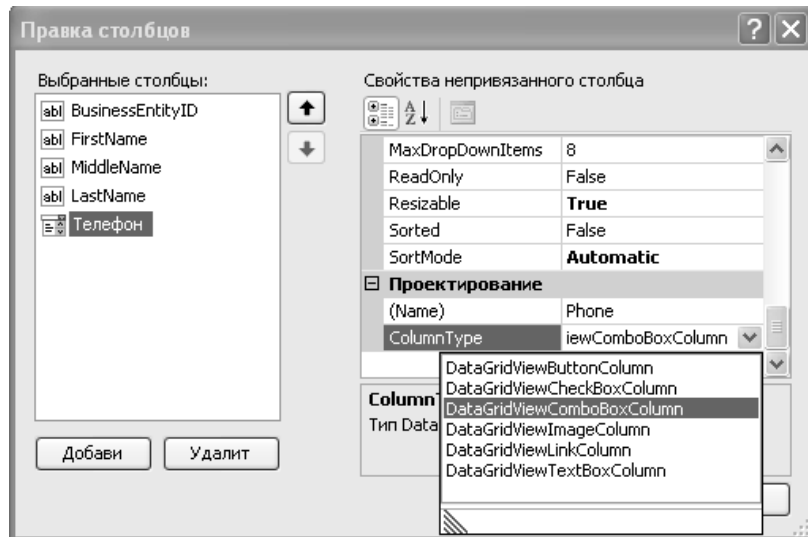


Рисунок 2.14. Установка свойства ColumnType.

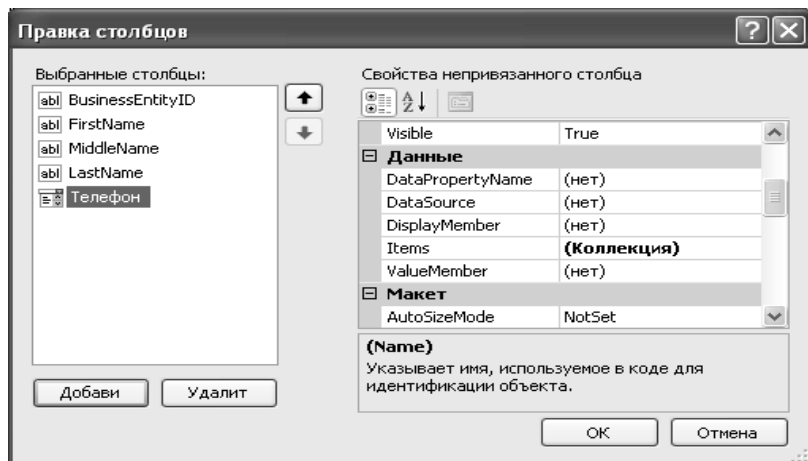


Рисунок 2.15. Свойства раздела Данные.

Теперь нужно правильно установить эти свойства.

- **DataSource.** Здесь необходимо выбрать нужную таблицу, входящую в НД. Для данного примера выбирается таблица PersonPhone (Другие источники данных → Источники данных проекта → adventureWorks2008DataSetBindingSource → PersonPhone).
- **DisplayMember.** Это свойство содержит имя поля, данные из которого будут отображаться в **DataGridView**. В данном случае это поле **Phone-Number**.
- **ValueMember.** Значение этого свойства соответствует внешнему ключу таблицы **PersonPhone (BusinessEntityID)**.
- **DataPropertyName.** В рассматриваемом примере значение этого свойства соответствует первичному ключу таблицы **Person (BusinessEntityID)**.
- **Items.** Данное свойство изменению не подлежит.

На рисунке 2.16 показаны все выбранные свойства для столбца **Phone**.

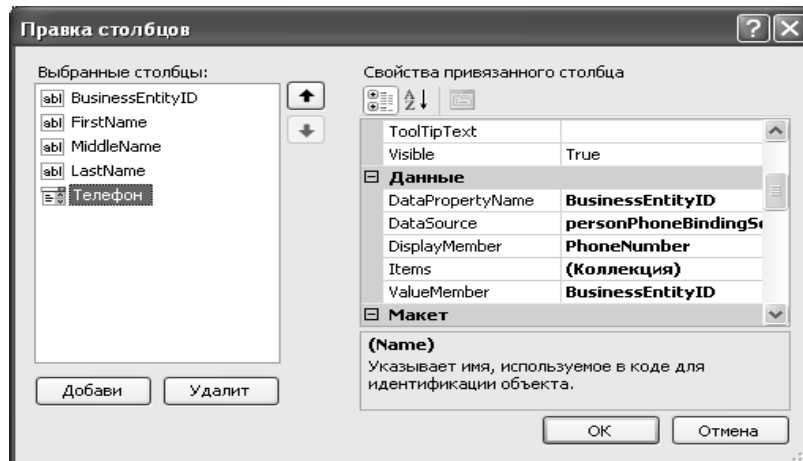


Рисунок 2.16. Свойства столбца Phone.

По нажатию кнопки **OK** на поддоне формы появятся новые компоненты **BindingSource** и **TableAdapter**, соответствующие новой таблице (рисунок 2.17).

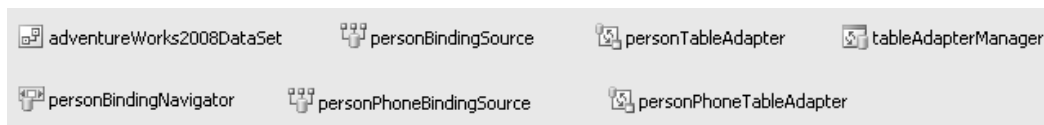


Рисунок 2.17. Новые компоненты для выбранной таблицы.

После запуска изменённого приложения можно убедиться, что данные из таблицы **PersonPhone** отображаются в столбце **Телефон** компонента **Data-GridView** (рисунок 2.18).

	BusinessEntityID	FirstName	MiddleName	LastName	Телефон
▶	295	Kim		Abercrombie	334-555-0137
	2170	Kim		Abercrombie	919-555-0100
	38	Kim	B	Abercrombie	208-555-0114
	211	Hazem	E	Abolrous	869-555-0125
	2357	Sam		Abolrous	567-555-0100
	285	Syed	E	Abraham	926-555-0182
	297	Humberto		Acevedo	599-555-0127
	291	Gustavo		Achong	398-555-0132
	299	Pilar		Ackerman	1 (11) 500 55...
	121	Pilar	G	Ackerman	577-555-0185
	16867	Aaron	B	Adams	417-555-0154
	16901	Adam		Adams	129-555-0195
	16724	Alex	C	Adams	346-555-0124
	10263	Alexandra	J	Adams	629-555-0159
	10312	Allison	L	Adams	1 (11) 500 55...
	10274	Amanda	P	Adams	592-555-0166

Рисунок 2.18. Отображение данных нескольких связанных таблиц.

2.3. Лабораторная работа № 2. Создание и модификация БД

Цель работы: знакомство со средой SQL Server Management Studio, создание с её помощью БД.

Содержание работы

1. Начало работы в среде SQL Server Management Studio.

Среда SQL Server Management Studio — это интегрированная среда для доступа, настройки, управления и администрирования всех компонентов SQL Server. Среда SQL Server Management Studio объединяет большое число графических средств с полнофункциональным редактором сценариев для доступа к SQL Server разработчиков и администраторов с любым опытом работы.

Среда SQL Server Management Studio поставляется вместе с SQL Server (кроме версии SQL Server Express).

Для запуска приложения необходимо выбрать **Пуск → Все программы → Microsoft SQL Server 2008 → SQL Server Management Studio**.

После запуска приложения необходимо осуществить соединение с SQL-сервером. В диалоговом окне **Connect to Server** можно выбрать тип сервера (по умолчанию Database Engine), имя сервера, тип аутентификации (варианты аутентификации выбираются на стадии установки SQL Server), имя пользователя и пароль. В случае выбора «Windows Authentication», имя пользователя и пароль будут недоступны для изменения, так как совпадают с таковыми в ОС MS Windows.

Для выполнения лабораторных работ выбор в данном диалоговом окне совпадает с предложенным по умолчанию, поэтому для соединения с сервером никаких изменений вносить не надо, достаточно нажать кнопку **Connect** (рисунок 3.1).

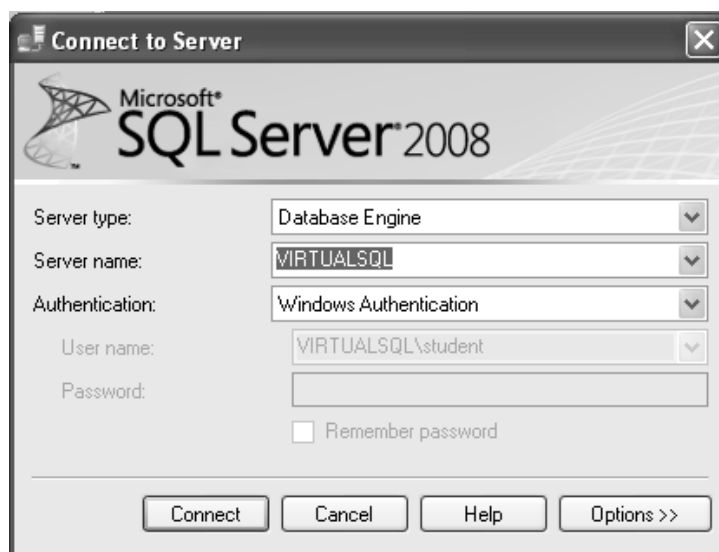


Рисунок 3.1. Диалоговое окно «Connect to Server»

SQL Server Management Studio содержит большое число компонентов. Часть из них может отображаться при начале работы, часть может оказаться видимой при выполнении каких-либо действий. Настройка комфортных условий для ра-

боты не содержит каких-либо хитростей. Для закрытия компонента достаточно нажать кнопку «х» нужного окна. При выборе в меню «View» того или иного компонента, он появляется на экране. Кнопка «Auto Hide» позволяет свернуть компонент в левую или правую часть экрана. Компоненты можно перемещать по экрану и закреплять в нужном положении.

Задание 1. Закрытие, скрывание, повторное открытие и перемещение компонентов.

- 1.1. Скрыть компонент «Registered Servers».
- 1.2. Выбрать для компонента «Object Explorer» вариант «Автоматически скрыть».
- 1.3. Восстановить окно «Registered Servers».
- 1.4. Поместить компонент «Object Explorer» в левой части окна, а компонент «Registered Servers» – в правой.
- 1.5. Совместить отображение компонентов в одно окно с закладками (рисунок 3.2).
- 1.6. Открыть два окна новых запросов (**New Query**) и настроить окно документов в режиме с закладками и с интерфейсом MDI (отдельные окна).
- 1.7. Отобразить страницу «Object Explorer Details».
- 1.8. Настроить параметры запуска (**Tools** → **Options** → **General**). Выбрать вариант «Open Object Explorer and new query».
- 1.9. Восстановить все значения по умолчанию.

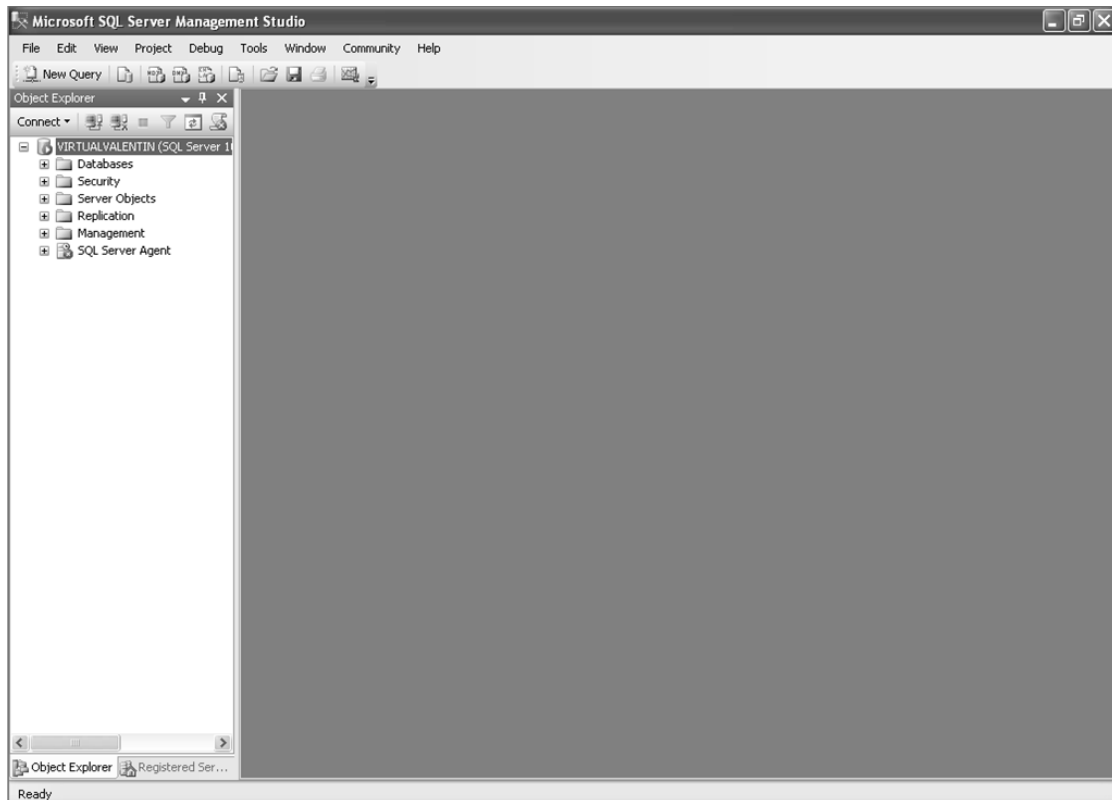


Рисунок 3.2. Окно SQL Server Management Studio
с двумя видимыми компонентами

2. Редактор запросов.

SQL Server Management Studio позволяет работать с БД используя Object Explorer, или создавая запросы на языке Transact-SQL (диалект SQL). При этом некоторые действия, совершённые в Object Explorer могут быть представлены в виде скрипта на языке Transact -SQL.

При работе с редактором запросов доступны следующие возможности:

- форматирование текст запроса (размещение в несколько строк, отступ);
- увеличение размеров окно редактора за счёт переключения в полноэкранный режим (**Shift+Alt+Enter**) или автоматического скрывтия всех других окон (**Window → Auto Hide All**);
- вставка комментария (два дефиса, «--») с помощью пункта **Edit → Advanced → Comment Selection**;
- преобразование комментария в текст (**Edit → Advanced → Uncomment Selection**);
- использование нескольких окон кода;
- запуск и отладка запроса.

Задание 2. Создание запроса.

Вся работа ведётся с БД **AdventureWorks2008**. Для того, чтобы избежать явного указания используемой БД в запросах, необходимо в дереве объектов компонента «Object Explorer» перейти к требуемой БД. В противном случае необходимо в начало текста запроса вставить строки

```
USE AdventureWorks2008;
```

```
GO
```

2.1. Открыть окно редактора запросов (**New Query**). В этом окне ввести текст запроса

```
SELECT * FROM Production.Product;
```

```
GO
```

2.2. Открыть новое окно. Создать второй запрос, который выбирает значения столбцов **BusinessEntityID**, **FirstName**, **MiddleName** и **LastName** из таблицы **Person.Person**.

```
-- Search for a contact
```

```
SELECT
```

```
BusinessEntityID,
```

```
FirstName,
```

```
MiddleName,
```

```
LastName
```

```
FROM Person.Person
```

```
WHERE LastName = 'Sanchez';
```

```
GO
```

2.3. Добавить для строк, содержащих имена столбцов, отступ.

2.4. Преобразовать строку «**WHERE LastName = 'Sanchez';**» в комментарий.

2.5. Сделать обратное преобразование комментария в текст.

2.6. Расположить окна редактора запросов горизонтально.

2.7. Расположить окна редактора запросов вертикально.

Замечание. Для запуска запроса на выполнение можно использовать пункт меню **Query** → **Execute**, клавишу **F5** или кнопку на панели запросов «**Execute**».

3. Использование шаблонов.

Среда Microsoft SQL Server Management Studio предлагает большое число шаблонов сценариев, содержащих инструкции Transact-SQL, предназначенных для решения многих типовых задач. Эти шаблоны содержат параметры, значения которых вводятся пользователем. Можно создавать собственные пользовательские шаблоны для поддержки сценариев, которые приходится писать чаще всего. Кроме того, можно реорганизовать дерево шаблонов, перемещая шаблоны или создавая новые папки для их хранения.

Для создания сценария на основе шаблона необходимо в меню **View** выбрать команду **Template Explorer**. Для внесения значений параметров используется пункт меню **Query** → **Specify Values for Template Parameters...**

Создание пользовательского шаблона может быть осуществлено по следующей схеме:

1. В обозревателе шаблонов раскрывается узел **SQL Server Templates**, в контекстном меню (правая кнопка мыши) элемента **Stored Procedure** выбирается пункт **New** → **Folder**.

2. Вводится имя новой папки шаблона.

3. Для вновь созданной папки в контекстном меню выбирается пункт **New Template**.

4. Вводится имя нового шаблона.

5. Для нового шаблона в контекстном меню выбирается команда **Edit**.

6. В редакторе запросов вводится сценарий создания хранимой процедуры, выполняющей необходимые действия. Необходимые параметры внедряются в сценарий с помощью следующего шаблона:

<field_name, field_type, default_value>,

где *field_name* – имя столбца, *field_type* – тип данных, *default_value* – значение по умолчанию.

7. Для сохранения шаблона используется команда **File** → **Save**.

Пример 1. Код сценария пользовательского шаблона с параметрами.

USE AdventureWorks2008

GO

IF EXISTS (

*SELECT **

FROM INFORMATION_SCHEMA.ROUTINES

WHERE SPECIFIC_NAME = 'WorkOrdersFor<product_name, nvarchar(50), name>')

DROP PROCEDURE dbo.WorkOrdersFor<product_name, nvarchar(50), name>
GO

CREATE PROCEDURE dbo.WorkOrdersFor<product_name, nvarchar(50), name>

AS

```
SELECT Name, WorkOrderID
FROM Production.WorkOrder AS WO
JOIN Production.Product AS Prod
ON WO.ProductID = Prod.ProductID
WHERE Name = '<product_name, nvarchar(50), name>';
GO
```

Задание 3. Создание пользовательского шаблона.

3.1. Создать шаблон, реализующий запрос к базе данных **AdventureWorks**, возвращающий список заказов на производство, имеющих дату выполнения меньше указанной (в качестве параметра).

Пояснение к заданию 3.1. Таблица с заказами - **Production.WorkOrder**, дата выполнения – столбец **DueDate**.

4. Использование проектов.

Отдельные сценарии можно объединять в проекты. Проекты сценариев включают данные о соединениях, необходимые для правильного выполнения сценариев, а также могут содержать файлы других типов, такие как вспомогательные текстовые файлы.

Для создания проекта необходимо выполнить следующую последовательность действий:

1. Создать проект. Для этого необходимо выбрать пункт меню **File → New → Project...**

2. В открывшемся диалоговом окне указывается тип действий (**SQL Server Scripts**), имя проекта, расположение файла проекта (**Location**) и действия с решением (**Solution**) (создать новое решение или добавить к имеющемуся, создать директорию для решения, имя решения) (рисунок 3.3.).

3. В обозревателе решений (**Solution Explorer**) можно настроить соединение с сервером (**Connections**), а также создать запросы (**Queries**).

Задание 4. Создание сценария.

4.1. Создать сценарий, работающий с БД **AdventureWorks2008** и реализующий запрос из задания 3.1.

Рассмотренные возможности SQL Server Management Studio дают лишь самое начальное представление об этом приложении. Если рассматривать изменения, которые произошли в SQL Server Management Studio со времени выхода SQL Server 2005, то их не очень много. Основные отличия состоят в реализации всплывающей подсказки, предлагающей возможные продолжения вводимого текста (IntelliSense), и возможности свернуть длинное выражение, щёлкнув по значку «минус» в его начале (для того, чтобы развернуть обратно, необходимо щёлкнуть по значку «плюс»).

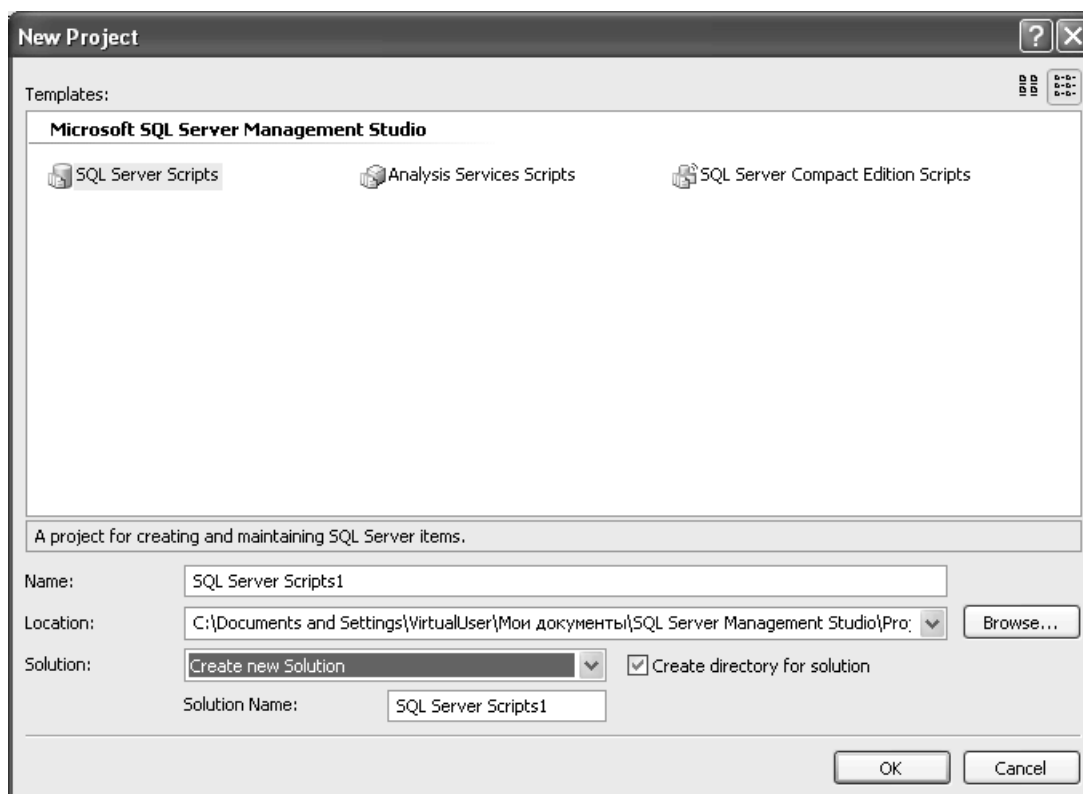


Рисунок 3.3. Окно создания нового проекта

5. Создание базы данных.

С точки зрения операционной системы БД в SQL Server 2008 представляет из себя (в простейшем случае) два файла – файл собственно БД, имеющий расширение *mdf*, и журнал транзакций – файл с расширением *ldf*. При создании БД могут быть установлены следующие параметры:

- имя БД (Database name) – обязательный параметр;
- владелец (Owner);
- начальный размер БД (Initial Size) – размер, который будет иметь mdf-файл в момент создания, измеряется в мегабайтах;
- авторасширение (Autogrowth) – позволяет администратору БД (АБД) управлять размером файла БД в процессе работы. Возможные значения:
 - разрешить/запретить авторасширение (Enable Autogrowth), в случае запрета изменять размер файла сможет только АБД, а отсутствие в нём свободного места вызовет ошибку при попытке внести новые данные в БД;
 - расширение файла (File Growth) – позволяет установить размер увеличения файла при исчерпании в нём свободного для данных места, указывается в процентах или мегабайтах;
 - максимальный размер файла (Maximum File Size) – позволяет задать максимальный размер, которого может достигнуть файл БД (Restricted File Growth), либо разрешить файлу увеличиваться в размерах неограниченно (Unrestricted File Growth);
- место расположения файла БД (путь);

- имя файла, содержащего БД (по умолчанию будет совпадать с логическим именем БД).

При создании журнала транзакций (создаётся одновременно с БД) могут быть установлены аналогичные свойства.

Следует заметить, что кроме указанных двух файлов, БД может содержать и другие, которые будут связаны с файловыми группами (filegroups) и внешними данными (filestream).

Для создания БД можно использовать визуальные средства SQL Server Management Studio или инструкцию Transact-SQL CREATE DATABASE.

Задание 5. Создание БД.

Используя визуальные средства SQL Server Management Studio создать БД со следующими свойствами:

- логическое имя БД – TestDB;
- начальный размер файла БД – 20 MB;
- авторасширение файла – без авторасширения;
- имя файла БД – TransactTestDB;
- место расположения – D:\Work\PM-31\Имя_Студента\.

Для журнала транзакций:

- начальный размер журнала – 5 MB;
- авторасширение файла – без авторасширения;
- имя файла журнала – TestDB_log;
- место расположения – D:\Work\PM-31\Имя_Студента\.

Сгенерировать скрипт для создания БД.

Задание 6. Модификация существующей БД.

Используя визуальные средства SQL Server Management Studio изменить БД TestDB:

- начальный размер файла БД – 5 MB;
- расширение файла – по 1 MB, итоговый размер не ограничен.

Внести изменения в свойства журнала транзакций:

- начальный размер журнала – 2 MB;
- расширение файла – по 5% до 2 GB.

Дополнительно: создать скрипт, выполняющий действия по изменению описания БД.

2.4. Лабораторная работа № 3. Создание таблиц

Цель работы: создание с помощью SQL Server Management Studio БД, набора таблиц в ней и внесение данных в БД.

Содержание работы

1. Типы данных.

Перед созданием таблиц необходимо определиться с типом данных для каждого поля каждой таблицы. SQL Server предоставляет большой выбор различных типов данных. Часть этих типов относится к стандартным, используемым в

ANSI-совместимых системах БД, часть используется только в MS SQL Server. Допускается создавать свои типы данных на основе имеющихся системных. В таблице 4.1. перечислены все системные типы данных, доступные в MS SQL Server 2008.

В MS SQL Server 2008 существует возможность создавать пользовательские (псевдонимные) типы данных. Они создаются на основе системных типов данных и служат для дополнительного уточнения типа данных с целью обеспечения совместимости при работе с общими элементами данных в различных таблицах или БД. Пользовательские типы данных определены для конкретной БД и должны иметь уникальные в пределах этой БД имена.

Таблица 4.1. Системные типы данных

Категория типа данных	Системные типы данных SQL Server 2008	ANSI-синоним	Число байтов
Целое число	int	integer	4
	bigint	—	8
	smallint	—	2
	tinyint	—	1
Точный числовой	decimal[(p[, s])]	dec	5-17
	numeric[(p[, s])]	—	5-17
Приблизительный числовой	float[(n)]	double precision, float[(n)]	4-8
	real	float(24)	4
Денежный	money	—	8
	smallmoney	—	4
Дата и время	datetime	—	8
	smalldatetime	—	4
	date	—	3
	time	—	3-5
	datetimeoffset	—	8-10
	datetime2	—	6-8
Символ не Юни-кода	char[(n)]	character[(n)]	0-8 000
	varchar[(n)]	char VARYING[(n)] character VARYING[(n)]	0-8 000
	varchar(max)	char VARYING(max) character VARYING(max)	0–2 ГБ (в строке или 16-разрядный указатель)
	text	—	0–2 ГБ (16-разрядный указатель)

Таблица 4.1. (Продолжение)

Категория типа данных	Системные типы данных SQL Server 2008	ANSI-синоним	Число байтов
Символ Юникода	nchar[(n)]	national char[(n)] national character[(n)]	0–8 000 (4 000 символов)
	nvarchar[(n)]	national char VARYING[(n)] national character VARYING[(n)]	0–8 000 (4 000 символов)
	nvarchar(max)	national char VARYING(max) national character VARYING(max)	0–2 ГБ (в строке или 16-разрядный указатель)
	ntext	—	0–2 ГБ (16-разрядный указатель)
Двоичный	binary[(n)]	—	0-8 000
	varbinary[(n)]	binary VARYING[(n)]	0-8 000
	varbinary(max)	binary VARYING(max)	0–2 ГБ (в строке или 16-разрядный указатель)
Изображение	image	—	0–2 ГБ (16-разрядный указатель)
Иерархический	hierarchyid	—	
Пространственный	geometry	—	
	geography	—	
Глобальный идентификатор	uniqueidentifier	—	16
XML	xml	—	0–2 ГБ
Специальный	bit	—	1
	cursor	—	0-8
	timestamp	rowversion	8
	sysname		256
	table		
	sql_variant		0–8016

Пользовательский тип данных может быть создан с помощью **Object Explorer** в среде SQL Server Management Studio, или с помощью инструкции CREATE TYPE языка Transact-SQL.

Следующий пример демонстрирует создание пользовательского типа данных с помощью инструкции CREATE TYPE.

Пример 1.

```
CREATE TYPE dbo.CountryCode
FROM char(2) NULL
```

Задание 1. Создание пользовательского типа данных.

1.1. Используя **Object Explorer** создать пользовательский тип данных, описывающий тип «Фамилия». Данный тип должен быть создан на основе системного типа varchar.

1.2. Используя инструкцию CREATE TYPE создать на основе varchar пользовательский тип данных «Имя».

Замечание 1. При создании пользовательского типа данных можно определить его возможность содержать значение NULL.

1.3. Удалить созданные пользовательские типы данных (DROP TYPE).

2. Создание таблиц.

Аналогично БД, таблицы можно создавать, используя среду SQL Server Management Studio, или с помощью языка Transact-SQL.

При создании таблицы необходимо задать имя таблицы, имена столбцов и типы данных столбцов. Имена столбцов должны быть уникальными для данной конкретной таблицы. Однако одно и то же имя столбца может использоваться в различных таблицах одной и той же БД. Для каждого столбца следует задать тип данных, возможность содержать значение NULL.

Замечание 2. Когда возможность столбца содержать значения NULL не задана, она определяется параметром сеанса или БД ANSI_NULL_DEFAULT. Когда значением этого параметра является ON, столбцы по умолчанию могут содержать значения NULL. Когда значением этого параметра является OFF, для столбцов по умолчанию используется признак NOT NULL. В SQL Server 2008 исходным значением параметра ANSI_NULL_DEFAULT является OFF.

При создании таблиц существуют следующие ограничения:

- более 2 млрд объектов на БД, включая таблицы;
- до 1024 столбцов на таблицу;
- до 8 060 байтов на строку (эта максимальная длина не применяется к столбцам, определенным с использованием спецификатора **max**.)

Существуют специальные типы столбцов:

- *Вычисляемые столбцы.* Вычисляемый столбец представляет собой виртуальный столбец, который не хранится в таблице в физическом виде. В SQL Server для расчёта значения такого столбца, когда в нём возникнет потребность, используется формула, определённая пользователем при создании этого столбца. Эта формула определяется с использованием других столбцов этой же таблицы.

- *Столбцы идентификаторов.* Свойство Identity может использоваться для создания столбцов, содержащих генерируемые системой последовательные значения, идентифицирующие каждый столбец, вставленный в таблицу.
- *Столбцы временных меток.* Столбцы, определенные с использованием типа данных timestamp, имеют значение по умолчанию, состоящее из автоматически генерируемой временной метки, гарантирующей уникальность внутри этой БД.
- *Столбцы uniqueidentifier.* Столбцы, определенные с использованием типа данных uniqueidentifier, могут применяться для хранения идентификаторов GUID (Globally Unique Identifier), гарантирующих уникальность в пределах БД. Значение для столбца uniqueidentifier может генерироваться с помощью функции NEWID языка Transact-SQL.

В MS SQL Server 2008 были добавлены новые типы данных, призванные как снять некоторые ограничения, существовавшие в предыдущих версиях (Date, Time, DateTimeOffset, DateTime2), так и расширяющие возможности по хранению и обработке неструктурированных, расширенных данных (Geometry, Geography, FileStream, Remote BLOB).

Работа с таблицами БД требует наличия возможности обратиться к какой-либо строке. В реляционной таблице строки не упорядочены и единственный способ добраться до требуемой записи – наличие некоторого объекта, позволяющего однозначно её идентифицировать. Таким объектом является ключ.

При работе с таблицами БД используются понятия «Первичный ключ» (Primary Key) и «Внешний ключ» (Foreign Key). Первичный ключ используется для однозначной идентификации строки в таблице, а внешний ключ реализует отношение между таблицами БД.

Замечание 3. Часто в качестве первичного ключа используется не набор столбцов, а идентификатор, генерируемый системой и не несущий смысловой нагрузки. Такой подход позволяет избежать громоздкости первичного ключа, а также гарантирует отсутствие проблем в будущем, связанных с возможной потерей по тем или иным причинам уникальности значений ключевого столбца.

Задание 2. Создание таблиц.

2.1. Используя среду SQL Server Management Studio создать в БД AdventureWorks2008 таблицу dbo.Table_1 со структурой, приведённой в таблице 4.2.

Таблица 4.2. Структура таблицы dbo.Table_1

Имя столбца	Тип данных	Разрешить значение NULL	Первичный ключ
DiscountId	int	нет	да, идентификатор
Amount	decimal (18,0)	нет	нет
DiscountName	nvarchar (50)	нет	нет
Description	nvarchar (MAX)	да	нет

2.2. Используя Transact-SQL создать в БД AdventureWorks2008 таблицу Sales.SpecialOffers со структурой, приведённой в таблице 4.3.

Таблица 4.3. Структура таблицы Sales.SpecialOffers

Имя столбца	Тип данных	Разрешить значение NULL	Первичный ключ
SpecialOfferID	int	нет	да, идентификатор
OfferName	nvarchar (25)	нет	нет
Description	nvarchar (MAX)	да	нет

2.3. Создать БД со всеми таблицами. Продумать и, возможно, доработать структуру, выявить взаимосвязи. При создании таблиц указать первичные и внешние ключи. Каждая таблица должна содержать 20-25 записей.

Вариант 1. Страховая компания.

Описание предметной области

Компания имеет различные филиалы по всей стране. Каждый филиал характеризуется названием, адресом и телефоном. Деятельность компании организована следующим образом: в компанию обращаются различные лица с целью заключения договора о страховании. В зависимости от принимаемых на страхование объектов и страхуемых рисков договор заключается по определенному виду страхования (например, страхование автотранспорта от угона, страхование домашнего имущества, добровольное медицинское страхование). При заключении договора фиксируется дата заключения, страховая сумма, вид страхования, тарифная ставка и филиал, в котором заключался договор. Нужно учесть, что договоры заключают страховые агенты. Помимо информации об агентах (фамилия, имя, отчество, адрес, телефон), нужно еще хранить филиал, в котором работают агенты. Кроме того, исходя из БД, нужно иметь возможность рассчитывать заработную плату агентам. Заработная плата составляет некоторый процент от страхового платежа (страховой платёж – это страховая сумма, умноженная на тарифную ставку). Процент зависит от вида страхования, по которому заключен договор.

Возможный набор сущностей:

- **Клиент** (Код клиента, Фамилия, Имя, Отчество, Адрес, Телефон).
- **Договоры** (Номер договора, Дата заключения, Страховая сумма, Тарифная ставка, Код филиала, Код вида страхования).

- **Вид страхования** (Код вида страхования, Наименование).
- **Филиал** (Код филиала, Наименование филиала, Адрес, Телефон).
- **Агент** (Фамилия, Имя, Отчество, Адрес, Телефон).

Вариант 2. Гостиница.

Описание предметной области

Гостиница предоставляет номера клиентам на определённый срок. Каждый номер характеризуется вместимостью, комфортностью (люкс, полулюкс, обычный) и ценой. Клиентами являются различные лица, о которых собирается определённая информация (фамилия, имя, отчество и некоторый комментарий). Сдача номера клиенту производится при наличии свободных мест в номерах, подходящих клиенту по указанным выше параметрам. При поселении фиксируется дата поселения. При выезде из гостиницы для каждого места запоминается дата освобождения. Необходимо не только хранить информацию по факту сдачи номера клиенту, но и осуществлять бронирование номеров. Кроме того, для постоянных клиентов, а также для определённых категорий клиентов (группа, студенты, пенсионеры) предусмотрена система скидок. Скидки могут суммироваться.

Возможный набор сущностей

- **Клиенты** (Код клиента, Фамилия, Имя, Отчество, Паспортные данные, Комментарий).
- **Номера** (Код номера, Номер, Количество человек, Комфортность, Цена).
- **Поселение** (Код поселения, Код клиента, Код номера, Дата поселения, Дата освобождения, Примечание).
- **Бронирование** (Код клиента, Код номера, Дата поселения).

Вариант 3. Реализация готовой продукции.

Описание предметной области

Деятельность компании организована следующим образом: компания торгует различными товарами. Каждый из этих товаров характеризуется наименованием, оптовой ценой, розничной ценой и справочной информацией. В компанию обращаются покупатели. Для каждого из них запоминаются стандартные данные (наименование, адрес, телефон, контактное лицо) и по каждой сделке составляется документ, содержащий кроме информации о покупателе количество купленного им товара и дату покупки. В зависимости от количества купленного товара могут предоставляться скидки от общей суммы сделки.

Возможный набор сущностей

- **Товары** (Код товара, Наименование, Оптовая цена, Розничная цена, Описание).
- **Покупатели** (Код покупателя, Телефон, Контактное лицо, Адрес).
- **Сделки** (Код сделки, Дата сделки, Код товара, Количество, Код покупателя, Признак оптовой продажи, Скидка).

Вариант 4. Ведение заказов.

Описание предметной области

Деятельность компании организована следующим образом: компания осуществляет оптовую торговлю различными товарами. Каждый из этих товаров характеризуется ценой, справочной информацией и признаком наличия или отсутствия доставки. В компанию обращаются заказчики. Для каждого из них запоминаются стандартные данные (наименование, адрес, телефон, контактное лицо) и по каждой сделке составляется документ, содержащий кроме информации о заказчике количество купленного им товара и дату покупки. Доставка разных товаров может производиться способами, различными по цене и скорости. Нужно хранить информацию о том, какими способами может осуществляться доставка каждого товара, и о том, какой вид доставки (а соответственно, и какую стоимость доставки) выбрал клиент при заключении сделки.

Возможный набор сущностей

- **Товары** (Код товара, Цена, Доставка, Описание).
- **Заказчики** (Код заказчика, Наименование, Адрес, Телефон, Контактное лицо).
- **Заказы** (Код заказа, Код заказчика, Код товара, Количество, Дата).
- **Доставка** (Код доставки, Код товара, Код заказа, Стоимость доставки, Срок доставки).

Вариант 5. Бюро по трудоустройству.

Описание предметной области

Деятельность бюро организована следующим образом: бюро готово искать работников для различных работодателей и вакансии для ищущих работу специалистов различного профиля. При обращении клиента-работодателя его стандартные данные (название, вид деятельности, адрес, телефон) фиксируются в БД. При обращении клиента-соискателя его стандартные данные (фамилия, имя, отчество, квалификация, профессия, иные данные) также фиксируются в БД. По каждому факту удовлетворения интересов обеих сторон составляется документ. В документе указываются соискатель, работодатель, должность и комиссионные (доход бюро).

Возможный набор сущностей

- **Работодатели** (Код работодателя, Название, Вид деятельности, Адрес, Телефон).
- **Соискатели** (Код соискателя, Фамилия, Имя, Отчество, Квалификация, Вид деятельности, Иные данные, Предполагаемый размер заработной платы).
- **Сделки** (Код соискателя, Код работодателя, Должность, Комиссионные).

Вариант 6. Нотариальная контора.

Описание предметной области

Деятельность нотариальной конторы организована следующим образом: фирма готова предоставить клиенту определённый комплекс услуг. При обращении клиента в контору его стандартные данные (название, вид деятельности, адрес, телефон) фиксируются в БД. По каждому факту оказания услуги клиенту составляется документ. В документе указываются услуга, сумма сделки, комиссионные (доход конторы), описание сделки. В рамках одной сделки клиенту может быть оказано несколько услуг. Для некоторых категорий клиентов (пенсионеры) и набора сделок могут быть предоставлены скидки на стоимость услуг.

Возможный набор сущностей

- **Клиенты** (Код клиента, Название, Вид деятельности, Адрес, Телефон).
- **Сделки** (Код сделки, Код клиента, Дата сделки, Код услуги, Сумма, Комиссионные, Описание).
- **Услуги** (Код услуги, Название, Описание).

Вариант 7. Фирма по продаже запчастей.

Описание предметной области

Рассматриваемая часть деятельности связана с работой с поставщиками. Фирма имеет определённый набор поставщиков, по каждому из которых известны название, адрес и телефон. У этих поставщиков приобретаются детали. Каждая деталь наряду с названием характеризуется артикулом и ценой. Цена может меняться от поставки к поставке. Разные поставщики могут поставлять одинаковые детали (один и тот же артикул). Каждый факт покупки запчастей у поставщика фиксируется в БД с указанием даты покупки и количества приобретённых деталей.

Возможный набор сущностей

- **Поставщики** (Код поставщика, Название, Адрес, Телефон).
- **Детали** (Код детали, Название, Артикул, Цена, Примечание).
- **Поставки** (Код поставщика, Код детали, Количество, Дата).

Вариант 8. Техническое обслуживание станков.

Описание предметной области

Предприятие занимается ремонтом станков и другого промышленного оборудования. Клиентами компании являются промышленные предприятия, оснащенные различным сложным оборудованием. В случае поломок оборудования они обращаются в рассматриваемую компанию. Ремонтные работы в компании организованы следующим образом: все станки проклассифицированы по странам-производителям, годам выпуска и маркам. Все виды ремонта отличаются названием, продолжительностью в днях, стоимостью. Исходя из этих данных, по каждому факту ремонта фиксируется вид станка и дата начала ремонта. Для каждого конкретного экземпляра оборудования необходимо хранить историю ремонтов.

Возможный набор сущностей

- **Виды станков** (Код вида станка, Страна, Год выпуска, Марка).

- **Станок** (Код вида станка, Серийный номер).
- **Виды ремонта** (Код ремонта, Название, Продолжительность, Стоимость, Примечания).
- **Ремонт** (Код вида станка, Код ремонта, Серийный номер, Дата начала, Примечания).

Вариант 9. Туристическая фирма.

Описание предметной области

Работа с клиентами в компании организована следующим образом: у каждого клиента собираются некоторые стандартные данные: фамилия, имя, отчество, адрес, телефон. Сотрудники выясняют у клиента, где он хотел бы отдыхать. При этом ему демонстрируются различные варианты, включающие страну проживания, особенности местного климата, имеющиеся отели разного класса. Наряду с этим обсуждается возможная длительность пребывания и стоимость путевки. В случае если удалось договориться и найти для клиента приемлемый вариант, регистрируется факт продажи путёвки, фиксируя дату отправления. Клиент может приобрести сразу несколько путёвок, возможно, различных по дате отправления и маршруту. Клиенту может быть предоставлена некоторая скидка, зависящая от длительности пребывания на отдыхе, даты отправления (сезона) и т.д.

Возможный набор сущностей

- **Маршруты** (Код маршрута, Страна, Климат, Длительность, Отель, Стоимость).
- **Путевки** (Код маршрута, Код клиента, Дата отправления, Количество, Скидка).
- **Клиенты** (Код клиента, Фамилия, Имя, Отчество, Адрес, Телефон).

Вариант 10. Грузовые перевозки.

Описание предметной области

Компания осуществляет перевозки по различным маршрутам. Для каждого маршрута существует название, примерное расстояние и некоторая оплата для водителя, зависящая от стажа. Перевозку могут осуществлять два водителя. Информация о водителях включает фамилию, имя, отчество и стаж. Для проведения расчётов хранится полная информация о перевозках (маршрут, водитель, даты отправки и прибытия). По факту некоторых перевозок водителям выплачивается премия.

Возможный набор сущностей

- **Маршруты** (Код маршрута, Название, Дальность, Количество дней в пути, Оплата).
- **Водители** (Код водителя, Фамилия, Имя, Отчество, Стаж).
- **Проделанная работа** (Код маршрута, Код водителя, Дата отправки, Дата возвращения, Премия).

Вариант 11. Учёт междугородних телефонных переговоров.

Описание предметной области

Абонентами компании являются юридические лица, имеющие телефонную точку, ИНН, расчетный счёт в банке. Стоимость переговоров зависит от города, в который осуществляется звонок, и времени суток (день, ночь). Каждый звонок абонента автоматически фиксируется в БД. При этом запоминаются город, дата, длительность разговора и время суток. Абонентам могут предоставляться скидки, зависящие от длительности разговора и города.

Возможный набор сущностей

- **Абоненты** (Код абонента, Номер телефона, ИНН, Адрес).
- **Города** (Код города, Название, Тариф дневной, Тариф ночной).
- **Переговоры** (Код переговоров, Код абонента, Код города, Дата, Количество минут, Время суток).

Вариант 12. Учет внутриофисных расходов.

Описание предметной области

Фирма состоит из отделов. Каждый отдел имеет название. В каждом отделе работает определённое количество сотрудников. Сотрудники могут осуществлять мелкие покупки для нужд фирмы в соответствии с видами расходов. Каждый вид расходов имеет название, некоторое описание и предельную сумму средств, которые могут быть потрачены сотрудниками конкретного отдела в месяц. Остатки неизрасходованных в текущем месяце средств переносятся на следующий период. При каждой покупке сотрудник оформляет документ, где указывает вид расхода, дату, сумму, отдел и свою фамилию.

Возможный набор сущностей

- **Отделы** (Код отдела, Название, Количество сотрудников).
- **Сотрудники** (Код сотрудника, Фамилия, Имя, Отчество, Код отдела).
- **Виды расходов** (Код вида, Название, Описание, Предельная норма).
- **Расходы** (Код расхода, Код вида, Код отдела, Код сотрудника, Сумма, Дата).

Вариант 13. Библиотека.

Описание предметной области

Библиотека оказывает платные услуги, выдавая напрокат некоторые книги, имеющиеся в небольшом количестве экземпляров. У каждой книги, выдаваемой в прокат, есть название, автор, жанр. В зависимости от ценности книги для неё определена залоговая стоимость (сумма, вносимая клиентом при взятии книги напрокат) и стоимость проката (сумма, которую клиент платит при возврате книги, получая назад залог), зависящая от срока проката. Существует система штрафов за вред, нанесённый книге. В библиотеку обращаются читатели. Все читатели регистрируются в картотеке, которая содержит стандартные анкетные данные (фамилия, имя, отчество, адрес, телефон). Каждый читатель может обращаться в библиотеку несколько раз. Все обращения читателей фик-

сируются, при этом по каждому факту выдачи книги запоминаются дата выдачи и ожидаемая дата возврата. Для читателей может быть предусмотрена система скидок.

Возможный набор сущностей

- **Книги** (Код книги, Название, Автор, Залоговая стоимость, Стоимость проката, Жанр).
- **Читатели** (Код читателя, Фамилия, Имя, Отчество, Адрес, Телефон).
- **Выданные книги** (Код книги, Код читателя, Дата выдачи, Дата возврата).
- **Штраф** (Код книги, Код читателя, Сумма штрафа, Комментарий).

Вариант 14. Прокат автомобилей.

Описание предметной области

В автопарк входит некоторое количество автомобилей различных марок, стоимостей и типов. Каждый автомобиль имеет свою стоимость проката. В пункт проката обращаются клиенты. Все клиенты проходят обязательную регистрацию, при которой о них собирается стандартная информация (фамилия, имя, отчество, адрес, телефон). Каждый клиент может обращаться в пункт проката несколько раз. Все обращения клиентов фиксируются, при этом по каждой сделке запоминаются дата выдачи и ожидаемая дата возврата. Предусмотрена система штрафов и скидок для клиентов.

Возможный набор сущностей

- **Автомобили** (Код автомобиля, Марка, Год выпуска, Стоимость, Стоимость проката, Тип).
- **Клиенты** (Код клиента, Фамилия, Имя, Отчество, Адрес, Телефон).
- **Выданные автомобили** (Код автомобиля, Код клиента, Дата выдачи, Дата возврата).
- **Штраф** (Код автомобиля, Код клиента, Сумма штрафа, Комментарий).

Вариант 15. Выдача банком кредитов.

Описание предметной области

Одним из существенных видов деятельности банка является выдача кредитов юридическим лицам. В зависимости от условий получения кредита, процентной ставки и срока возврата все кредитные операции делятся на несколько основных видов. Каждый из этих видов имеет своё название. Кредит может получить клиент, при регистрации предоставивший следующие сведения: название, вид собственности, адрес, телефон, контактное лицо. Каждый факт выдачи кредита регистрируется банком, при этом фиксируются сумма кредита, клиент и дата выдачи. Требуется хранить информацию о динамике возврата кредита, дате фактического возврата денег, штрафах за просроченные платежи по кредиту.

Возможный набор сущностей

- **Виды кредитов** (Код вида, Название, Условия получения, Ставка, Срок).
- **Клиенты** (Код клиента, Название, Вид собственности, Адрес, Телефон, Контактное лицо).

- **Кредиты** (Код вида, Код клиента, Сумма, Дата выдачи).
- **Платежи по кредиту** (Код кредитного договора, Дата платежа, Сумма, Штраф).

Вариант 16. Инвестирование свободных средств.

Описание предметной области

Инвестиционная компания занимается вложением денежных средств в ценные бумаги. Клиенты компании – предприятия, которые доверяют управлять их свободными денежными средствами на определённый период. Необходимо выбрать вид ценных бумаг, которые позволят получить прибыль и компании, и клиенту. При работе с клиентом весьма существенной является информация о предприятии – название, вид собственности, адрес и телефон. Требуется хранить историю котировок ценных бумаг.

Возможный набор сущностей

- **Ценные бумаги** (Код ценной бумаги, Минимальная сумма сделки, Рейтинг, Доходность за прошлый год, Дополнительная информация).
- **Инвестиции** (Код инвестиции, Код ценной бумаги, Код клиента, Котировка, Дата покупки, Дата продажи).
- **Клиенты** (Код клиента, Название, Вид собственности, Адрес, Телефон).

Вариант 17. Занятость актёров театра.

Описание предметной области

Каждый год театр осуществляет постановку различных спектаклей. Каждый спектакль имеет определённый бюджет. Для участия в конкретных постановках в определённых ролях привлекаются актёры. При этом на одну и ту же роль может привлекаться несколько актёров. С каждым из актёров заключается персональный контракт, включающий некоторую базовую сумму вознаграждения и премиальные по итогам реально отыгранных спектаклей. Каждый из актёров имеет некоторый стаж работы, некоторые из них удостоены различных наград и званий.

Возможный набор сущностей

- **Актёры** (Код актёра, Фамилия, Имя, Отчество, Звание, Стаж).
- **Спектакли** (Код спектакля, Название, Год постановки, Бюджет).
- **Занятость актёров в спектакле** (Код актёра, Код спектакля, Роль, Стоимость годового контракта).
- **Участие в конкретном спектакле** (Код актёра, Код спектакля, Дата спектакля, Премии).

Вариант 18. Платная поликлиника.

Описание предметной области

В поликлинике работают врачи различных специальностей, имеющие разную квалификацию. Каждый день в поликлинику обращаются больные. Все они проходят обязательную регистрацию, при которой в БД заносятся стандартные анкетные данные (фамилия, имя, отчество, год рождения). Каждый

больной может обращаться в поликлинику несколько раз, нуждаясь в различной медицинской помощи. При этом пациент в рамках одного обращения может обследоваться и проходить лечение у разных специалистов. Все обращения больных фиксируются, при этом устанавливается диагноз, определяется стоимость лечения, запоминается дата обращения.

Возможный набор сущностей

- **Врачи** (Код врача, Фамилия, Имя, Отчество, Специальность, Категория).
- **Пациенты** (Код пациента, Фамилия, Имя, Отчество, Год рождения).
- **Обращения** (Код обращения, Код врача, Код пациента, Дата обращения, Диагноз, Стоимость лечения).

Вариант 19. Анализ динамики показателей финансовой отчётности предприятий.

Описание предметной области

В структуру холдинга входят несколько предприятий. Каждое предприятие имеет стандартные характеристики (название, реквизиты, телефон, контактное лицо). Работа предприятия может быть оценена следующим образом: в начале каждого отчётного периода на основе финансовой отчётности вычисляется по неким формулам определённый набор показателей. Некоторые показатели могут содержать данные в различных валютах (рубли, доллары, евро). Важность показателей характеризуется некоторыми числовыми константами. Значение каждого показателя измеряется в некоторой системе единиц.

Возможный набор сущностей

- **Показатели** (Код показателя, Название, Важность, Единица измерения).
- **Предприятия** (Код предприятия, Название, Банковские реквизиты, Телефон, Контактное лицо).
- **Динамика показателей** (Код показателя, Код предприятия, Дата, Значение).

Вариант 20. Учёт телекомпанией стоимости прошедшей в эфире рекламы

Описание предметной области

Работа построена следующим образом: заказчики просят разместить свою рекламу в определённой передаче в определённый день. Каждый рекламный ролик имеет определённую продолжительность. Для каждой организации-заказчика известны банковские реквизиты, телефон и контактное лицо для проведения переговоров. Передачи имеют определённый рейтинг. Стоимость минуты рекламы в каждой конкретной передаче известна (определяется коммерческой службой исходя из рейтинга передачи и прочих соображений). Договор на показ рекламного ролика заключается агентом, чья зарплата составляет некоторый процент от стоимости рекламы, прошедшей в эфире.

Возможный набор сущностей

- **Передачи** (Код передачи, Название, Рейтинг, Стоимость минуты).
- **Реклама** (Код рекламы, Код передачи, Код заказчика, Дата, Длительность в минутах).

- **Заказчики** (Код заказчика, Название, Банковские реквизиты, Телефон, Контактное лицо).
- **Агент** (Код агента, Фамилия, Имя, Отчество, Код рекламы, Код передачи, Код заказчика).

Вариант 21. Интернет-магазин

Описание предметной области

Работа компании организована следующим образом: на Интернет-сайте представлены (выставлены на продажу) некоторые товары. Каждый из них имеет некоторое название, цену и единицу измерения (штуки, килограммы, литры). Для проведения исследований и оптимизации работы магазина производится сбор данных с клиентов. При этом собираются стандартные анкетные данные, телефон и адрес электронной почты для связи. В случае приобретения товаров на сумму свыше 5 000 руб. клиент переходит в категорию постоянных и получает скидку на каждую покупку в размере 2%. По каждому факту продажи автоматически фиксируется клиент, товары, количество, дата продажи, дата доставки. В рамках одной продажи клиент может приобрести несколько товаров.

Возможный набор сущностей

- **Товары** (Код товара, Название, Цена, Единица измерения).
- **Клиенты** (Код клиента, Фамилия, Имя, Отчество, Адрес, Телефон, e-mail, Признак постоянного клиента).
- **Продажи** (Код продажи, Код товара, Код клиента, Дата продажи, Дата доставки, Количество).

Вариант 22. Ювелирная мастерская.

Описание предметной области

Ювелирная мастерская осуществляет изготовление ювелирных изделий для частных лиц на заказ. Работа ведётся с определёнными материалами (платина, золото, серебро, различные драгоценные камни и т.д.). При обращении потенциального клиента определяется, какое именно изделие ему необходимо. Все изготавливаемые изделия принадлежат к некоторому типу (серьги, кольца, броши, браслеты), выполнены из одного или нескольких определённых материалов, имеют некоторый вес для каждого использованного материала и цену (включающую стоимость материалов и работы). Постоянным клиентам мастерская предоставляет скидки.

Возможный набор сущностей

- **Изделия** (Код изделия, Название, Тип, Код материала, Вес, Цена).
- **Материалы** (Код материала, Название, Цена за грамм).
- **Продажи** (Код изделия, Дата продажи, Фамилия покупателя, Имя покупателя, Отчество покупателя, Скидка).

Вариант 23. Парикмахерская.

Описание предметной области

Парикмахерская обслуживает клиентов в соответствии с их пожеланиями и некоторым каталогом различных видов стрижки. Так, для каждой стрижки определены название, принадлежность полу (мужская, женская), стоимость работы. С целью составления базы клиентов при наличии согласия с их стороны в БД вносятся их анкетные данные (фамилия, имя, отчество). Начиная с пятой стрижки, клиент переходит в категорию постоянных и получает скидку в 3% при каждой последующей стрижке. После того как закончена очередная работа, документом фиксируются стрижка, клиент и дата производства работ.

Возможный набор сущностей

- **Стрижки** (Код стрижки, Название, Пол, Стоимость).
- **Клиенты** (Код клиента, Фамилия, Имя, Отчество, Пол, Признак постоянного клиента).
- **Работа** (Код работы, Код стрижки, Код клиента, Дата).

Вариант 24. Сдача в аренду торговых площадей.

Описание предметной области

Работа торгового центра построена следующим образом: в результате планирования определено некоторое количество торговых точек в пределах здания, которые могут сдаваться в аренду. Для каждой из торговых точек важными данными являются этаж, площадь, наличие кондиционера и стоимость аренды в день. Со всех потенциальных клиентов собираются стандартные данные (название, адрес, телефон, реквизиты, контактное лицо). Потенциальному клиенту показываются имеющиеся свободные площади. При достижении соглашения оформляется договор, фиксируя в БД торговую точку, клиента, период (срок) аренды. Клиент может арендовать сразу несколько торговых точек. Необходимо собирать информацию о ежемесячных платежах, поступающих от арендаторов.

Возможный набор сущностей

- **Торговые точки** (Код торговой точки, Этаж, Площадь, Наличие кондиционера, Стоимость аренды в день).
- **Клиенты** (Код клиента, Название, Реквизиты, Адрес, Телефон, Контактное лицо).
- **Аренда** (Код аренды, Код торговой точки, Код клиента, Дата начала, Дата окончания).
- **Платежи** (Код торговой точки, Код клиента, Платёжный период, Сумма платежа).

2.5. Лабораторная работа № 4. Выполнение простых запросов

Цель работы: создание запросов.

Содержание работы

1. Простые запросы.

Стандарт ISO SQL не поддерживает формальные термины *отношение*, *атрибут* и *кортеж*, вместо них применяются термины *таблица*, *столбец* и *строка*. Ещё одной особенностью стандарта SQL является отступление от строгой поддержки определений реляционной модели данных. Например, в языке SQL допускается наличие повторяющихся строк в таблице, созданной в результате выполнения операции SELECT, для столбцов устанавливается определённая последовательность, строки могут быть отсортированы в некотором порядке, определённом пользователем.

Оператор SQL состоит из зарезервированных слов, а также из слов, определяемых пользователем. Зарезервированные слова являются постоянной частью языка SQL и имеют определённое значение. Зарезервированные слова следует записывать именно так, как указано в стандарте, их нельзя разбивать на части для переноса на другую строку. Слова, определяемые пользователем, задаются самим пользователем (в соответствии с определёнными синтаксическими правилами) и представляют имена различных объектов БД (таблиц, столбцов, представлений, индексов и т.д.). Слова в операторе размещаются в соответствии с установленными синтаксическими правилами.

Замечание 1. Многие диалекты требуют задания в конце оператора некоторого символа («;»), обозначающего окончание его текста, хотя в стандарте такое требование отсутствует.

Язык SQL имеет свободный формат записи, но существуют неформальные правила:

- каждая конструкция в операторе должна начинаться с новой строки;
- начало каждой конструкции должно быть обозначено таким же отступом, что и начало других конструкций операторов;
- если конструкция состоит из нескольких частей, каждая из них должна начинаться с новой строки с некоторым отступом относительно начала конструкции, что будет указывать на их подчинённость.

Для определения формата операторов SQL используется расширенная форма системы обозначений Бэкуса-Наура:

- прописные буквы используются для записи зарезервированных слов;
- строчные буквы используются для записи слов, определяемых пользователем;
- вертикальная черта («|») указывает на необходимость выбора одного из нескольких приведенных значений (например, $a \mid b \mid c$);
- фигурные скобки определяют обязательный элемент ($\{a\}$);
- квадратные скобки определяют необязательный элемент ($[a]$);

- многоточие (...) указывает необязательную возможность повторения конструкции от нуля до нескольких раз ({a|b} [, c...]).

Замечание 2. Литералы представляют собой константы, которые используются в операторах SQL. Все нечисловые значения данных всегда должны заключаться в одинарные кавычки, а числовые – нет.

Оператор SELECT – выборка и отображение данных одной или более таблиц БД.

Общий формат оператора SELECT:

```
SELECT [DISTINCT | ALL] { * | [columnExpression [AS newName]] [, ...] }
FROM TableName [alias] [, ...]
[WHERE condition]
[GROUP BY columnList] [HAVING condition]
[ORDER BY columnList],
```

где columnExpression – имя столбца или выражение из нескольких имён, TableName – имя существующей в БД таблицы (или представления), alias – сокращение, устанавливаемое для имени таблицы TableName.

Приведённый формат оператора SELECT является самым базовым и не включает все возможные параметры и опции, доступные в различных диалектах. Более полный (хотя и не самый полный) вариант оператора SELECT можно рассмотреть на примере MS SQL Server 2008.

Пример 1. Синтаксис оператора SELECT в MS SQL Server 2008.

```
<SELECT statement> ::=
    [WITH <common_table_expression> [,...n]]
    <query_expression>
    [ ORDER BY { order_by_expression | column_position [ ASC | DESC ] } [, ...n]
] ]
    [ COMPUTE
        { { AVG | COUNT | MAX | MIN | SUM } ( expression ) } [, ...n ]
        [ BY expression [, ...n ] ]
    ]
    [ <FOR Clause> ]
    [ OPTION ( <query_hint> [, ...n ] ) ]
<query_expression> ::=
    { <query_specification> | ( <query_expression> ) }
    [ { UNION [ ALL ] | EXCEPT | INTERSECT }
        <query_specification> | ( <query_expression> ) [, ...n ]
    ]
<query_specification> ::=
    SELECT [ ALL | DISTINCT ]
    [ TOP ( expression ) [ PERCENT ] [ WITH TIES ] ]
    <select_list>
    [ INTO new_table ]
    [ FROM { <table_source> } [, ...n ] ]
    [ WHERE <search_condition> ]
```

[<GROUP BY>]

[HAVING < search_condition >]

Обработка элементов оператора SELECT выполняется в следующей последовательности:

- FROM – определяются имена используемой таблицы (таблиц);
- WHERE – выполняется фильтрация строк объекта в соответствии с заданными условиями;
- GROUP BY – образуются группы строк, имеющих одно и то же значение в указанном столбце;
- HAVING – фильтруются группы строк объекта в соответствии с указанным условием;
- SELECT – устанавливается, какие столбцы должны присутствовать в выходных данных;
- ORDER BY – определяется упорядоченность результатов выполнения оператора.

Замечание 3. Порядок конструкций в операторе SELECT не может быть изменён.

Замечание 4. Только SELECT и FROM являются обязательными.

Замечание 5. Операция выборки с помощью SELECT является замкнутой (результат запроса также является таблицей).

Пример 2. Вывести все данные о заказчиках, поставщиках или сотрудниках из таблицы Person.Person.

Вариант 1.

```
SELECT [BusinessEntityID]
      ,[PersonType]
      ,[NameStyle]
      ,[Title]
      ,[FirstName]
      ,[MiddleName]
      ,[LastName]
      ,[Suffix]
      ,[EmailPromotion]
      ,[AdditionalContactInfo]
      ,[Demographics]
      ,[rowguid]
      ,[ModifiedDate]
```

```
FROM [AdventureWorks2008].[Person].[Person]
```

Замечание 6. В случае, если требуется включить в результат запроса все столбцы таблицы, можно воспользоваться более компактным способом записи, указав вместо перечисления имён столбцов символ «*». В этом случае поля будут расположены в том же порядке, что и в исходной таблице.

Пример 2. Вариант 2.

```
SELECT *
FROM [AdventureWorks2008].[Person].[Person]
```

Пример 3. Вывести личные данные (FirstName, MiddleName, LastName) из таблицы Person.Person.

```
SELECT [FirstName]
      ,[MiddleName]
      ,[LastName]
FROM [AdventureWorks2008].[Person].[Person]
```

Допускается использование вычисляемых полей. Для создания вычисляемого поля в списке SELECT следует указать некоторое выражение языка SQL. Могут применяться операции сложения, вычитания, умножения и деления, могут использоваться круглые скобки. Тип используемых столбцов должен быть числовым. При этом по умолчанию имя такого столбца определяется конкретным диалектом. В MS SQL Server 2008 у такого поля не будет никакого имени (пример 4, рисунок 5.1). Стандарт ISO предоставляет возможность явного указания желаемого имени с использованием конструкции «AS newName», где newName – имя вычисляемого столбца, заданное пользователем (пример 5, рисунок 5.2).

Пример 4. Вывести данные о товаре (Name) и цене со скидкой 10% (ListPrice*0.9) из таблицы Production.Product.

```
SELECT [Name],[ListPrice]*0.90
FROM [AdventureWorks2008].[Production].[Product]
```

	Name	(No column name)
1	LL Mountain Seat Assembly	120.006000
2	ML Mountain Seat Assembly	132.426000
3	HL Mountain Seat Assembly	177.228000
4	LL Road Seat Assembly	120.006000
5	ML Road Seat Assembly	132.426000
6	HL Road Seat Assembly	177.228000
7	LL Touring Seat Assembly	120.006000
8	ML Touring Seat Assembly	132.426000

Рисунок 5.1. Использование вычисляемых полей

Пример 5. Вывести данные о товаре (Name) и цене со скидкой 10% (ListPrice*0.9) из таблицы Production.Product. Присвоить вычисляемому полю имя «NewPrice».

```
SELECT [Name],[ListPrice]*0.90 AS NewPrice
FROM [AdventureWorks2008].[Production].[Product]
```

	Name	NewPrice
1	LL Mountain Seat Assembly	120.006000
2	ML Mountain Seat Assembly	132.426000
3	HL Mountain Seat Assembly	177.228000
4	LL Road Seat Assembly	120.006000
5	ML Road Seat Assembly	132.426000
6	HL Road Seat Assembly	177.228000
7	LL Touring Seat Assembly	120.006000
8	ML Touring Seat Assembly	132.426000

Рисунок 5.2. Использование именованных вычисляемых полей.

Замечание 7. Давать новые имена можно любым столбцам, а не только вычисляемым. При этом в MS SQL Server 2008 использование ключевого слова AS не является обязательным.

Пример 6. Вывести данные о наименовании (Name) и цене (ListPrice) товара из таблицы Production.Product. Включить в результирующую таблицу только те товары, для которых указана ненулевая цена. Товары расположить в порядке убывания цены. Дать полям имена «Наименование» и «Цена» (рисунок 5.3).

```
SELECT [Name] AS [Наименование],[ListPrice] AS [Цена]
FROM [AdventureWorks2008].[Production].[Product]
WHERE [ListPrice]>0.009
ORDER BY [ListPrice] DESC
```

	Наименование	Цена
1	Road-150 Red, 62	3578,27
2	Road-150 Red, 44	3578,27
3	Road-150 Red, 48	3578,27
4	Road-150 Red, 52	3578,27
5	Road-150 Red, 56	3578,27
6	Mountain-100 Silver, 38	3399,99
7	Mountain-100 Silver, 42	3399,99
8	Mountain-100 Silver, 44	3399,99

Рисунок 5.3. Результат выполнения запроса из примера 6

Стандарт ISO содержит определение пяти агрегирующих функций:

- COUNT – возвращает количество значений в указанном столбце;
- SUM – возвращает сумму значений в указанном столбце;
- AVG – возвращает усреднённое значение в указанном столбце;
- MIN – возвращает минимальное значение в указанном столбце;
- MAX – возвращает максимальное значение в указанном столбце.

Все эти функции оперируют со значениями в единственном столбце и возвращают единственное значение. При работе исключаются все пустые значения. COUNT, MIN, MAX применимы и для нечисловых полей. COUNT(*) подсчитывает все строки в таблице независимо от того, являются ли значения пустыми, повторяющимися и т.д. Если требуется исключить повторяющиеся строки, то необходимо использовать ключевое слово DISTINCT. Агрегирующие

функции могут использоваться только в списке выборки SELECT и в конструкции HAVING. Если SELECT содержит агрегирующую функцию, а в тексте запроса отсутствует конструкция GROUP BY, то ни один из элементов списка выборки SELECT не может включать каких-либо ссылок на столбцы, за исключением случая, когда этот столбец используется как параметр агрегирующей функции.

Пример 7. Данный пример содержит ошибку, так как в списке выборки кроме агрегирующей функции используется поле «Name» при отсутствии конструкции GROUP BY.

```
SELECT [Name] AS [Наименование],AVG([ListPrice]) AS [Цена]
FROM [AdventureWorks2008].[Production].[Product]
WHERE [ListPrice]>0.009
```

Пример 8. Вывести среднюю цену по всем продуктам.

```
SELECT AVG([ListPrice]) AS [Цена]
FROM [AdventureWorks2008].[Production].[Product]
WHERE [ListPrice]>0.009
```

Пример 9. Вывести данные о средней цене за товары в зависимости от даты начала продаж.

```
SELECT [SellStartDate],AVG([ListPrice]) AS MiddlePrice
FROM [AdventureWorks2008].[Production].[Product]
WHERE [ListPrice]>0.009
GROUP BY [SellStartDate]
```

Задание 1. Создание простых запросов.

Для выполнения данного задания используется БД AdventureWorks2008 и MS SQL Server 2008.

1.1. Вывести всю информацию обо всех продуктах из таблицы Production.Product. Результат отсортировать по наименованию продукта (поле Name).

1.2. Вывести информацию о наименовании (Name), номере (ProductNumber) и цене (Listprice) всех продуктов из таблицы Production.Product. Результат отсортировать по наименованию продукта. Присвоить столбцам результирующей таблицы заголовки на русском языке.

1.3. Вывести информацию о наименовании, номере и цене всех продуктов из таблицы Production.Product, для которых линейки продуктов начинаются символом R и для которых длительность изготовления не превышает 4 дней. Результат отсортировать по наименованию продукта.

1.4. Вывести информацию о должностях сотрудников (JobTitle) компании из таблицы HumanResources.Employee. Результат отсортировать по алфавиту. Удалить из результирующей таблицы все повторяющиеся строки.

1.5. Найти общий объем продаж (LineTotal) для каждого заказа из таблицы Sales.SalesOrderDetail. Результат отсортировать по коду заказа (SalesOrderID).

1.6. Вычислить средние цены и объёмы продаж из таблицы Sales.SalesOrderDetail, сгруппированные по коду продукта (ProductID) и идентификатору специального предложения (SpecialOfferID).

1.7. Вывести информацию о наименовании, цене всех продуктов и коде модели (ProductModelID) из таблицы Production.Product, для которых цена превышает \$1000. Результат отсортировать по наименованию продукта.

1.8. Вывести информацию о коде модели продукта (ProductModelID) и средней цене по этому коду из таблицы Production.Product. Рассматриваются только продукты, цена на которые превышает \$1000. Результат отсортировать по коду модели.

1.9. Вывести из таблицы Sales.SalesOrderDetail информацию о среднем количестве приобретаемых продуктов (OrderQty) и стоимости покупки (OrderQty * UnitPrice). Результат сгруппировать по стоимости и вывести в порядке убывания.

1.10. Вывести из таблицы Sales.SalesOrderDetail информацию о коде и средней цене проданных продуктов, для которых количество в одном заказе превышает 10 штук. Результат отсортировать по средней цене.

1.11. Вывести из таблицы Sales.SalesOrderDetail информацию о коде проданных продуктов, для которых среднее количество в заказах превышает 5 штук. Результат отсортировать по коду продуктов.

1.12. Вывести из таблицы Production.Product информацию обо всех продуктах, в наименовании которых встречается слово «ball».

1.13. Вывести из таблицы Person.PersonPhone информацию обо всех телефонных номерах с региональным кодом 415.

1.14. Вывести из таблицы Person.Person информацию о сотрудниках с именем Cheryl или Sheryl. Результат отсортировать по фамилии.

1.15. Вывести из таблицы Person.Person информацию о сотрудниках с фамилией Zheng или Zhang. Результат отсортировать по фамилии и имени.

1.16. Посчитать количество сотрудников, носящих фамилии Zheng и Zhang.

Пояснение к заданиям 1.14-1.16.

В этих заданиях предполагается, что требуется отобрать тех сотрудников, чьи фамилии отличаются на несколько символов, расположенных в заданных позициях. Для случая английского языка в заданиях предложено найти всех сотрудников с созвучными фамилиями.

1.17. Вывести из таблицы Production.Product информацию о продуктах с наименованиями Blade, Crown Race или Spokes.

1.18. Вывести из таблицы Person.Person список сотрудников в виде XML-документа.

Пояснение к заданию 1.18.

Предложение FOR XML используется в инструкции Transact-SQL SELECT для извлечения данных в виде XML вместо строк и столбцов.

Синтаксис конструкции FOR XML.

FOR XML

```

{
  { RAW [ ( 'ElementName' ) ] | AUTO }
  [
    <CommonDirectives>
    [ , { XMLDATA | XMLSCHEMA [ ( 'TargetNameSpaceURI' ) ] } ]
    [ , ELEMENTS [ XSINIL | ABSENT ]
  ]
| EXPLICIT
  [
    <CommonDirectives>
    [ , XMLDATA ]
  ]
| PATH [ ( 'ElementName' ) ]
  [
    <CommonDirectives>
    [ , ELEMENTS [ XSINIL | ABSENT ] ]
  ]
}

<CommonDirectives> ::=
[ , BINARY BASE64 ]
[ , TYPE ]
[ , ROOT [ ( 'RootName' ) ] ]

```

Аргументы.

XML – задаёт возврат результатов запроса в виде XML-документа. Должен быть задан один из следующих режимов XML: RAW, AUTO, EXPLICIT.

RAW [(*ElementName*)] – получает результат запроса и преобразует каждую строку результирующего набора в элемент XML, для которого в качестве тега используется общий идентификатор <row />. Дополнительно можно задать имя для элемента строки. Для результирующего выхода в формате XML в качестве элементов, создаваемых для каждой строки, используются определённые *ElementName*.

AUTO – возвращает результаты запроса в виде простого вложенного дерева XML. Каждая таблица предложения FROM, для которой в предложении SELECT приведён хотя бы один столбец, отображается как элемент XML. Столбцы, перечисленные в предложении SELECT, сопоставлены с соответствующими атрибутами элемента.

EXPLICIT – задаёт явное определение формы результирующего XML-дерева. При использовании данного режима запросы должны записываться таким образом, чтобы дополнительные сведения о вложениях могли быть заданы явно.

XMLDATA – возвращает встроенную XDR-схему, не добавляя корневой элемент к результату. При задании параметра XMLDATA XDR-схема добавляется к документу.

XMLSCHEMA [('*TargetNameSpaceURI*')] – возвращает встроенную XSD-схему. При задании указанной директивы, возвращающей заданное пространство имён схемы, дополнительно можно задать URI целевого пространства имён.

ELEMENTS – задаёт возврат столбцов в виде вложенных элементов. В противном случае столбцы будут сопоставлены с XML-атрибутами. Данный параметр поддерживается только в режимах RAW, AUTO и PATH.

XSINIL – задаёт создание элемента с атрибутом **xsi:nil**, установленного в значение **True**, для столбцов со значениями **NULL**. Данный параметр может быть указан только в директиве ELEMENTS.

ABSENT – указывает, что соответствующие XML-элементы для столбцов со значениями **NULL** к XML-результату не добавляются. Данный параметр используется только с директивой ELEMENTS.

PATH [('*ElementName*')] – создаёт упаковщик элементов <строки> для каждой строки в результирующем наборе. Для упаковщика элементов <строки> можно дополнительно задать имя элемента. При задании пустой строки, например FOR XML PATH (''), упаковщик элементов не создаётся. Использование директивы PATH даёт более простой способ написания запросов, чем написание запросов с помощью директивы EXPLICIT.

BINARY BASE64 – задаёт возврат двоичных данных запросом в двоичном зашифрованном формате base64. При извлечении двоичных данных с использованием режимов RAW и EXPLICIT необходимо указывать этот параметр. В режиме AUTO это указывается по умолчанию.

TYPE – задаёт следующий формат выдаваемых запросом данных: тип **xml**.

ROOT [('*RootName*')] – задаёт добавление единичного элемента высшего уровня к результирующему XML-документу. Дополнительно можно указать имя корневого элемента, который необходимо сформировать. Если имя корневого элемента не задано, то добавляется <корневой> элемент по умолчанию.

1.19. Вывести из таблицы Person.Person список сотрудников, чья фамилия начинается на «G» в виде XML-документа. Для создания XML-документа использовать аргументы AUTO, TYPE, XMLSCHEMA, ELEMENTS XSINIL.

1.20. Составить несколько (не менее пяти) однотабличных запросов на выборку данных к БД своего варианта, созданной на лабораторной работе № 3. Запросы должны содержать:

- просмотр таблицы целиком;
- использование условий-ограничений (проверка на равенство/неравенство значений, проверка на соответствие шаблону, использование арифметических и логических выражений);
- сортировку выборки;
- ограничение выборки;

- функции для работы со строковыми, числовыми и временными значениями (конкатенация, математические функции, функции для вычисления различных значений по дате и времени);
- получение статистических и обобщающих данных из таблиц (подсчёт числа значений в столбце, подсчёт количества по группам свойств, использование агрегирующих функций, группировка по условию).

2.6. Лабораторная работа № 5. Выполнение многотабличных запросов

Цель работы: создание многотабличных запросов.

Содержание работы

1. Многотабличные запросы.

Задание 1. Создание многотабличных запросов.

Для выполнения данного задания используется БД AdventureWorks2008 и MS SQL Server 2008.

1.1. Вывести информацию (фамилия, имя) о кандидатах на должность, принятых на работу. Использовать таблицы Person.Person (поля FirstName, LastName и BusinessEntityID) и HumanResources.JobCandidate (поле BusinessEntityID).

1.2. Вывести информацию (фамилия, имя, дата поступления на работу) о сотрудниках, проработавших в компании менее десяти лет. Использовать таблицы Person.Person (поля FirstName, LastName и BusinessEntityID) и HumanResources.Employee (поля BusinessEntityID и HireDate). Результат отсортировать по дате поступления на работу.

1.3. Вывести информацию (фамилия, имя, дата поступления на работу) о кандидатах на должность, принятых на работу. Использовать таблицы Person.Person (поля FirstName, LastName и BusinessEntityID), HumanResources.JobCandidate (поле BusinessEntityID) и HumanResources.Employee (поля BusinessEntityID и HireDate).

1.4. Вывести информацию обо всех заказах (дата и номер документа) с указанием сотрудников (фамилия, имя), если есть, отвечавших за эти заказы. Использовать таблицы Person.Person и Sales.SalesOrderHeader. Результат отсортировать по дате документа. Необходимые поля: BusinessEntityID, FirstName и LastName (таблица Person.Person), OrderDate, SalesOrderNumber и SalesPersonID (таблица Sales.SalesOrderHeader).

1.5. Вывести информацию (фамилию, имя) обо всех сотрудниках компании с указанием заказов, за которые они отвечали, а также дату и номер документа о заказе. Если сотрудник не отвечал ни за какие заказы, то он всё равно должен быть включён в этот список. Использовать таблицы Person.Person, HumanResources.Employee и Sales.SalesOrderHeader. Результат отсортировать по коду сотрудника (BusinessEntityID). Необходимые поля: BusinessEntityID (таблицы Person.Person, HumanResources.Employee), FirstName и LastName (таблица Per-

son.Person), OrderDate, SalesOrderNumber и SalesPersonID (таблица Sales.SalesOrderHeader).

1.6. Вывести информацию о продукте (ProductID, Name), находившемся в производстве в июле 2001 года. Использовать таблицы Production.Product и Production.WorkOrder. Создать запросы двух типов: с использованием операции пересечения и без неё.

1.7. Вывести информацию о продукте (ProductID, Name), не находившемся в производстве в июле 2001 года. Использовать таблицы Production.Product и Production.WorkOrder. Создать запросы двух типов: с использованием операции вычитания и без неё.

1.8. Вывести информацию обо всех доходах с продаж и скидках по каждому продукту. Использовать таблицы Production.Product (поля ProductID и Name) и Sales.SalesOrderDetail (поля ProductID, OrderQty, UnitPrice и UnitPrice-Discount). Создать запросы двух видов: с использованием внешнего соединения и без него.

1.9. Вывести наименование товара (таблица Production.Product, поля ProductModelID и Name), которому в таблице Production.ProductModel (поля ProductModelID и Name) соответствует наименование 'Long-sleeve logo jersey' (кофта с длинными рукавами, с эмблемой). Составить запросы с использованием ключевого слова EXISTS и ключевого слова IN.

1.10. Вывести информацию (фамилия, имя из таблицы Person.Person) о сотрудниках, для которых значение премии в таблице Sales.SalesPerson составляет 5 000 (поле Bonus).

1.11. Вывести информацию (фамилия, имя из таблицы Person.Person и должность из таблицы HumanResources.Employee) о сотрудниках, для которых значение премии в таблице Sales.SalesPerson составляет 5 000 (поле Bonus).

1.12. Вывести информацию о моделях продуктов (таблица Production.Product, поля ProductModelID и ListPrice), для которых максимальная цена в каталоге превышает среднюю цену по нему на 2% и больше.

1.13. Вывести информацию (фамилия, имя из таблицы Person.Person) о сотрудниках, продававших товар 'BK-M68B-42' (таблица Production.Product, поле ProductNumber). Необходимо задействовать таблицы Sales.SalesOrderHeader (поле SalesOrderID) и Sales.SalesOrderDetail (поле ProductID).

1.14. Вывести информацию о подразделениях компании, в которых есть сотрудники, занимающие должность «Scheduling Assistant», с указанием количества таких должностей. Использовать таблицы HumanResources.Department, HumanResources.Employee, HumanResources.EmployeeDepartmentHistory. При создании запроса учесть значение поля EndDate (NULL) в таблице HumanResources.EmployeeDepartmentHistory.

1.15. Составить несколько (не менее пяти) многотабличных запросов на выборку данных к созданной на лабораторной работе № 3 по своему варианту БД.

2.7. Лабораторная работа № 6. Дополнительные возможности MS SQL

Цель работы: создание секционированных таблиц, моментальных снимков, представлений, индексов.

Содержание работы

1. Создание секционированных таблиц.

В секционированной таблице данные разделены горизонтально на блоки, которые могут быть распределены между различными файловыми группами. Деление происходит на основе диапазона значений конкретного столбца. Типичным примером использования служит таблица, содержащая данные о заказах на продажу, которая разделена на секции на основании даты заказа. Заказы, размещённые в текущем году, помещены в одну секцию, заказы прошлого года – в другую, а все остальные, более ранние, – в третью.

Такой подход позволяет, во-первых, улучшить управляемость таблиц. Так как данные размещены в разных файловых группах и могут иметь различную актуальность, то появляется возможность применения различных стратегий резервного копирования для разных файловых групп. Разные файловые группы можно размещать на разных носителях, что позволяет подбирать наиболее подходящее хранилище данных с учетом требований, связанных с обеспечением к ним соответствующего доступа – более оперативные данные могут быть размещены на высокоскоростных носителях (RAID-массивы), а менее востребованные данные можно разместить в сжатых файловых группах. Секционирование можно применять и для индексов, что даёт все вышеуказанные преимущества, а кроме того, снижает фрагментацию индекса, так как перестройке подвергается только та его часть, которая связана с секцией оперативных данных.

Во-вторых, работа с секционированными таблицами повышает производительность. Это достигается за счёт более быстрого выполнения операций поиска по индексу, соединения таблиц. Кроме того, уменьшается риск возникновения блокировок и взаимоблокировок, что также благотворно влияет на общую производительность.

Создание секционированной таблицы требует некоторых подготовительных действий. Первое, необязательное действие, состоит в создании файловых групп, если этого не было сделано ранее, с целью распределения секций между этими группами.

Второе действие – создание функции секционирования. Функция секционирования определяет тип данных ключа, использованного для секционирования данных и граничных значений для каждой секции. Количество секций, определяемое функцией секционирования, всегда на единицу больше, чем количество граничных значений, определяемых этой функцией, так как первая секция будет создана под данные, ключевое значение которых по величине меньше значения первой границы, а последняя секция будет содержать данные, ключевое значение которых по величине больше значения последней границы.

Замечание 1. Функция секционирования определяет критерий распределения строк таблицы по секциям, основываясь на конкретном типе данных и конкретных диапазонах значений, и не является специфичной для конкретной таблицы, что позволяет использовать её многократно.

Синтаксис оператора создания функции секционирования в MS SQL Server 2008.

```
CREATE PARTITION FUNCTION partition_function_name ( input_parameter_type )
AS RANGE [ LEFT | RIGHT ]
FOR VALUES ( [ boundary_value [ ,...n ] ] )
[ ; ]
```

Аргументы.

partition_function_name – имя функции секционирования. Имена функций секционирования должны быть уникальными внутри БД и соответствовать правилам для идентификаторов.

input_parameter_type – тип данных столбца, используемого для секционирования. В качестве столбцов секционирования могут использоваться данные любого типа, кроме *text*, *ntext*, *image*, *xml*, *timestamp*, *varchar(max)*, *nvarchar(max)*, *varbinary(max)*, типов данных-псевдонимов, а также определяемых пользователем типов данных CLR.

boundary_value – задаёт граничные значения для каждой секции секционированной таблицы или индекса, в которой используется аргумент *partition_function_name*. Если параметр *boundary_value* не задан, функция секционирования сопоставляет всю таблицу или индекс одной секции, используя аргумент *partition_function_name*. Может использоваться только один столбец секционирования, заданный с помощью инструкции CREATE TABLE или CREATE INDEX.

...n – задаёт до 999 значений, указанных в аргументе *boundary_value*. Количество созданных секций равно $n + 1$. Значения могут отображаться не по порядку. Если значения выведены не по порядку, компонент Database Engine сортирует их, создаёт функцию и выдает предупреждение о том, что значения выданы не по порядку. В случае наличия в параметре *n* повторяющихся значений компонент Database Engine выдает ошибку.

LEFT | RIGHT – указывает, к какой области интервала значений принадлежит аргумент *boundary_value* [,...n] (к левой или правой) для случая, когда значения интервалов были отсортированы компонентом Database Engine по возрастанию слева направо. Если значение аргумента не указано, то по умолчанию принимается LEFT.

Замечание 2. Функции секционирования могут применяться только внутри БД, в которой они были созданы. Функции секционирования располагаются в отдельном от других функций пространстве имён внутри БД.

Замечание 3. Все строки, которым соответствуют значения NULL столбца секционирования, располагаются в самой левой секции, кроме случая, когда за-

дано пустое граничное значение и параметр RIGHT. В данном случае самая левая секция является пустой, и в неё помещаются значения NULL.

Пример 1. Создание функции секционирования для столбца данных типа *int*.

```
CREATE PARTITION FUNCTION myRangePF1 (int)
AS RANGE LEFT FOR VALUES (1, 100, 1000);
```

В примере 1 создаётся функция, которая будет делить таблицу на секции, относя значения, точно совпадающие с граничным, к левой секции (RANGE LEFT). Следовательно, строка, в которой значение поля, по которому идёт секционирование, равно 1, будет отнесена к самой левой секции. В случае задания значения «RANGE RIGHT», такая строка относится к правой секции.

Пример 2. Создание функции секционирования для столбца данных типа *datetime*.

```
CREATE PARTITION FUNCTION myRangePF2 (datetime)
AS RANGE RIGHT FOR VALUES ('20090201', '20090301', '20090401',
'20090501', '20090601', '20090701', '20090801', '20090901', '20091001', '20091101',
'20091201');
```

Замечание 4. Использование формата «ууууymmdd» для представления даты является наиболее универсальным, так как не зависит от языка сеанса.

В примере 2 создаётся функция, которая делит таблицу на секции в соответствии с месяцами года (самая левая секция будет содержать даты до 1 февраля 2009 года, а самая правая – даты, начиная с 1 декабря 2009 года).

Пример 3. Создание функции секционирования для столбца данных типа *char*.

```
CREATE PARTITION FUNCTION myRangePF3 (char(20))
AS RANGE RIGHT FOR VALUES ('AB', 'BCD', 'DEF');
```

Замечание 5. Для функции секционирования поддерживаются операторы ALTER PARTITION FUNCTION и DROP PARTITION FUNCTION. Оператор ALTER PARTITION FUNCTION позволяет разбить имеющуюся секцию на две или объединить две имеющихся секции в одну.

Следующий этап построения секционированных таблиц – создание схемы секционирования. Схема секционирования сопоставляет секции, определённые в конкретной функции секционирования, с файловыми группами, в которых эти секции будут физически храниться. Все секции могут быть сопоставлены одной и той же файловой группе; некоторые или же все секции могут быть сопоставлены различным файловым группам. Подобно функции секционирования схема не является специфичной для конкретной таблицы, что позволяет использовать её многократно.

При создании схемы секционирования можно дополнительно задать файловую группу, которая будет использоваться в случае добавления другой секции к функции секционирования. Она носит название «*Следующая файловая группа*».

Синтаксис оператора создания функции секционирования в MS SQL Server 2008.

```
CREATE PARTITION SCHEME partition_scheme_name
```

```
AS PARTITION partition_function_name
[ ALL ] TO ( { file_group_name | [ PRIMARY ] } [ ,...n ] )
[ ; ]
```

Аргументы.

partition_scheme_name – имя схемы секционирования. Имена схемы секционирования должны быть уникальными в пределах БД и должны соответствовать правилам для идентификаторов.

partition_function_name – имя функции секционирования, использующей схему секционирования. Секции, созданные функцией секционирования, сопоставляются с файловыми группами, заданными в схеме секционирования. Аргумент *partition_function_name* уже должен существовать в БД.

ALL – указывает, что все секции сопоставляются с файловой группой, определяемой аргументом *file_group_name*, или с первичной файловой группой, если указывается [PRIMARY]. Если указывается ALL, то может быть указано только одно значение аргумента *file_group_name*.

file_group_name | [PRIMARY] [,...n] – указывает имена файловых групп, содержащих секции, указываемые аргументом *partition_function_name*. Если указывается [PRIMARY], секция сохраняется в первичной файловой группе. Если указывается ALL, то может быть указано только одно значение аргумента *file_group_name*. Секции назначаются файловым группам, начиная с секции 1, в том порядке, в котором файловые группы перечисляются в [,...n]. Одно и то же имя *file_group_name* может быть указано в [,...n] несколько раз. Если значение *n* не достаточно для количества секций, указываемого в аргументе *partition_function_name*, CREATE PARTITION SCHEME завершается с ошибкой.

Если аргумент *partition_function_name* формирует меньше секций, чем количество файловых групп, первая неназначенная файловая группа отмечается как NEXT USED (следующая файловая группа) и информационное сообщение выводит наименование файловой группы NEXT USED. Если указывается параметр ALL, единственный аргумент *file_group_name* сохраняет свое свойство NEXT USED для аргумента *partition_function_name*. Файловая группа NEXT USED получит дополнительную секцию, если такая секция будет создана инструкцией ALTER PARTITION FUNCTION. Чтобы создать дополнительные неназначенные файловые группы, которые должны содержать новые секции, используется инструкция ALTER PARTITION SCHEME.

В примерах 4-7 вначале идёт оператор создания функции секционирования (CREATE PARTITION FUNCTION), а потом – оператор создания схемы секционирования (CREATE PARTITION SCHEME).

Пример 4. Создание схемы секционирования, которая всем секциям ставит в соответствие разные файловые группы.

```
CREATE PARTITION FUNCTION myRangePF1 (int)
AS RANGE LEFT FOR VALUES (1, 100, 1000);
GO
```

```
CREATE PARTITION SCHEME myRangePS1
AS PARTITION myRangePF1
TO (test1fg, test2fg, test3fg, test4fg);
```

Пример 5. Создание схемы секционирования, ставящей в соответствие несколько секций одной файловой группе.

```
CREATE PARTITION FUNCTION myRangePF2 (int)
AS RANGE LEFT FOR VALUES (1, 100, 1000);
GO
CREATE PARTITION SCHEME myRangePS2
AS PARTITION myRangePF2
TO ( test1fg, test1fg, test1fg, test2fg );
```

Пример 6. Создание схемы секционирования, сопоставляющей все секции с одной файловой группой.

```
CREATE PARTITION FUNCTION myRangePF3 (int)
AS RANGE LEFT FOR VALUES (1, 100, 1000);
GO
CREATE PARTITION SCHEME myRangePS3
AS PARTITION myRangePF3
ALL TO ( test1fg );
```

Пример 7. Создание схемы секционирования, указывающей файловую группу «NEXT USED».

```
CREATE PARTITION FUNCTION myRangePF4 (int)
AS RANGE LEFT FOR VALUES (1, 100, 1000);
GO
CREATE PARTITION SCHEME myRangePS4
AS PARTITION myRangePF4
TO (test1fg, test2fg, test3fg, test4fg, test5fg);
```

Замечание 6. Для схемы секционирования поддерживаются операторы ALTER PARTITION SCHEME и DROP PARTITION SCHEME. Оператор ALTER PARTITION SCHEME добавляет файловую группу в схему секционирования или изменяет обозначение файловой группы NEXT USED для данной схемы секционирования.

После создания функции и схемы секционирования, можно приступить к последнему этапу – созданию секционированной таблицы. Чтобы секционировать таблицу в процессе её создания, необходимо в операторе CREATE TABLE указать:

- схему секционирования, которая будет использована таблицей для сопоставления секций с файловыми группами;
- столбец, по которому таблица должна быть секционирована (столбец секционирования); столбец секционирования должен совпадать с указанным в функции секционирования, которая используется схемой секционирования, по типу данных, длине и точности.

Пример 8. Создание секционированной таблицы.

```

CREATE PARTITION FUNCTION myRangePF1 (int)
AS RANGE LEFT FOR VALUES (1, 100, 1000);
GO
CREATE PARTITION SCHEME myRangePS1
AS PARTITION myRangePF1
TO (test1fg, test2fg, test3fg, test4fg);
GO
CREATE TABLE PartitionTable (col1 int, col2 char(10))
ON myRangePS1 (col1);
GO

```

2. Создание моментальных снимков.

Моментальный снимок БД является статичным, доступным только для чтения представлением БД (*базы данных-источника*). В базе данных-источнике может находиться несколько моментальных снимков, которые всегда принадлежат тому же экземпляру сервера, что и БД. Каждый моментальный снимок БД согласуется на уровне транзакций с базой данных-источником в момент создания моментального снимка. Моментальный снимок существует до тех пор, пока он не будет явно удалён владельцем БД.

Моментальные снимки можно использовать для составления отчётов. Кроме того, при возникновении пользовательской ошибки в базе данных-источнике, эту БД можно восстановить до состояния, в котором она находилась на момент создания моментального снимка. Произойдёт только потеря изменений в БД, произведённых после создания моментального снимка.

Замечание 7. Моментальные снимки БД доступны только в выпуске SQL Server Enterprise.

Ограничения на создание моментальных снимков.

Моментальный снимок БД должен располагаться на том же сервере, что и база данных-источник. В отношении снимков БД также действуют следующие ограничения:

- моментальные снимки не могут создаваться для БД model, master и tempdb;
- моментальные снимки БД не подлежат резервному копированию и восстановлению;
- моментальный снимок БД не может быть присоединён или отсоединён;
- моментальные снимки БД не могут создаваться в разделах FAT32 и в неразмеченных разделах;
- перед удалением БД необходимо удалить все её моментальные снимки;
- SQL Server Management Studio не поддерживает создание моментальных снимков, поэтому снимки БД можно создавать только с помощью Transact-SQL.

Используемая в SQL Server 2008 (и SQL Server 2005) технология копирования при записи позволяет делать моментальные снимки БД, не создавая её полную копию. Первоначально снимок БД пуст и физически представляет собой

разреженные файлы NTFS — файлы, для которых место на диске выделяется только тогда, когда это необходимо. При первом обновлении какой-либо страницы в базе данных-источнике первоначальный образ этой страницы копируется в моментальный снимок БД. Если страница никогда не изменяется, она никогда и не копируется.

SQL Server делает копии целых страниц данных, даже если на них обновляется лишь одна строка. Использование страниц вместо строк повышает эффективность обработки. Производительность операций чтения и записи для целой страницы практически та же, что и для отдельной строки. На странице может содержаться большое число строк, и даже при обновлении сразу нескольких строк в SQL Server достаточно выполнить только одну операцию копирования.

При обращении к моментальному снимку БД пользователь увидит в снимке копию страницы, только если она изменялась с момента создания снимка. В противном случае пользователь будет перенаправлен на соответствующую страницу в базе данных-источнике. Такое перенаправление осуществляется прозрачно для пользователя.

Для создания моментального снимка БД используется стандартная конструкция CREATE DATABASE с помощью предложения AS SNAPSHOT OF.

Пример 9. Создание моментального снимка БД.

```
CREATE DATABASE AdventureWorks_Snapshot1200 ON
(NAME = N'AdventureWorks2008_Data', FILENAME =
N'C:\Program Files\Microsoft SQL Server\MSSQL.1\MSSQL\Data\AW_1200.ss')
AS SNAPSHOT OF AdventureWorks2008;
```

Моментальный снимок БД может быть удалён с помощью оператора

DROP DATABASE *Snapshot_Name*;

Snapshot_Name – имя удаляемого моментального снимка.

3. Создание представлений.

Представление – это виртуальная таблица, чьё содержимое определяется запросом. Как и реальная таблица, представление состоит из набора именованных столбцов и строк данных. Представление существует в БД в виде хранимого набора значений данных, только если оно проиндексировано. Используются строки и столбцы данных из таблиц, на которые выполняются ссылки в запросе, определяющем представление, и они создаются динамически при ссылке на представление. Таблицы, запрашиваемые в представлении, называются базовыми таблицами.

Типичные примеры представлений:

- подмножество строк (горизонтальное представление) или столбцов базовой таблицы (вертикальное представление);
- объединение одной или нескольких базовых таблиц;
- соединение одной или нескольких базовых таблиц;
- статистическая сводка базовой таблицы;
- подмножество другого представления или некоторая комбинация представлений и базовых таблиц.

Представления обычно используются в следующих случаях:

- сокрытие от пользователя ненужных или конфиденциальных данных;
- упрощение работы с данными;
- повышение безопасности благодаря предоставлению пользователям возможности обращаться к данным через представление без получения разрешения на прямой доступ к основным базовым таблицам представления;
- обеспечение обратной совместимости путём определения эмуляции представлением таблицы, которая существовала ранее, но схема которой была изменена;
- определение набора данных, которые пользователь может экспортировать и импортировать в SQL Server;
- обеспечение консолидированного представления секционированных данных, т. е. сходных данных, которые хранятся в разных таблицах.

Синтаксис оператора создания представления в MS SQL Server 2008.

```
CREATE VIEW [ schema_name . ] view_name [ (column [ ,...n ] ) ]
[ WITH <view_attribute> [ ,...n ] ]
AS select_statement
[ WITH CHECK OPTION ] [ ; ]
```

<view_attribute> ::=

```
{
    [ ENCRYPTION ]
    [ SCHEMABINDING ]
    [ VIEW_METADATA ] }
```

Аргументы.

schema_name – имя схемы, которой принадлежит представление.

view_name – имя представления. Имена представлений должны соответствовать требованиям, предъявляемым к идентификаторам.

column – имя, которое будет иметь столбец в представлении. Имя столбца требуется только в тех случаях, когда столбец формируется на основе арифметического выражения, функции или константы, если два или более столбцов могут по иной причине получить одинаковые имена или если столбцу представления назначается имя, отличное от имени столбца, от которого он произведён. Назначать столбцам имена можно также в инструкции SELECT. Если аргумент *column* не указан, столбцам представления назначаются такие же имена, которые имеют столбцы в инструкции SELECT.

AS – определяет действия, которые должны быть выполнены в представлении.

select_statement – инструкция SELECT, которая определяет представление. В этой инструкции можно указывать более одной таблицы и другие представления. Представление не обязательно является простым подмножеством строк и столбцов одной конкретной таблицы. С помощью предложения SELECT

можно создавать представление, использующее более одной таблицы, или другие представления любой степени сложности.

Предложения SELECT, используемые в определении представления, не могут включать следующие элементы:

- предложения COMPUTE и COMPUTE BY;
- предложение ORDER BY, если только в списке выбора инструкции SELECT нет также предложения TOP;
- ключевое слово INTO;
- предложение OPTION;
- ссылку на временную таблицу или табличную переменную.

В аргументе *select_statement* можно использовать функции и множественные инструкции SELECT, разделённые оператором UNION или UNION ALL.

CHECK OPTION – обеспечивает соответствие всех выполняемых для представления инструкций модификации данных критериям, заданным при помощи аргумента *select_statement*. Если строка изменяется посредством представления, предложение WITH CHECK OPTION гарантирует, что после фиксации изменений доступ к данным из представления сохранится.

ENCRYPTION – выполняет шифрование элементов представления sys.syscomments, содержащего текст инструкции CREATE VIEW. Использование предложения WITH ENCRYPTION предотвращает публикацию представления в рамках репликации SQL Server.

SCHEMABINDING – привязывает представление к схеме базовой таблицы или таблиц. Если аргумент SCHEMABINDING указан, нельзя изменить базовую таблицу или таблицы таким способом, который может повлиять на определение представления. При использовании аргумента SCHEMABINDING инструкция *select_statement* должна включать двухкомпонентные (*schema.object*) имена таблиц, представлений или пользовательских функций, упоминаемых в предложении. Все указанные в инструкции объекты должны находиться в одной БД.

VIEW_METADATA – указывает, что экземпляр SQL Server возвратит в API-интерфейсы DB-Library, ODBC и OLE DB сведения метаданных о представлении вместо базовой таблицы или таблиц, когда метаданные режима обзора затребованы для запроса, который ссылается на представление. Метаданные режима обзора — это дополнительные метаданные, которые экземпляр SQL Server возвращает вышеназванным клиентским API-интерфейсам. Эти метаданные позволяют клиентским API-интерфейсам реализовывать обновляемые клиентские курсоры. Метаданные режима обзора содержат сведения о базовой таблице, которой принадлежат столбцы в результирующем наборе.

Для представлений, созданных с применением предложения VIEW_METADATA, метаданные режима обзора возвращают имя представления, а не имена базовых таблиц при описании столбцов из представления в результирующем наборе.

Пример 10. Использование простой инструкции CREATE VIEW.

```
USE AdventureWorks2008;
GO
IF OBJECT_ID ('hiredate_view', 'V') IS NOT NULL
DROP VIEW hiredate_view ;
GO
CREATE VIEW hiredate_view
AS
SELECT c.FirstName, c.LastName, c.BusinessEntityID, e.HireDate
FROM HumanResources.Employee e JOIN Person.Person c on e.BusinessEntityID =
c.BusinessEntityID ;
GO
```

Замечание 8. Для представлений определены операторы ALTER VIEW и DROP VIEW.

4. Создание индексов.

Индекс – это набор страниц, связанный с таблицей или представлением и используемый для ускорения получения строк из таблицы или обеспечения уникальности. Индекс содержит ключи, построенные по одному или нескольким столбцам таблицы.

Индекс в БД можно сравнить с предметным указателем в книге. Его назначение – поиск нужной информации без необходимости последовательного просмотра таблицы.

Создание индекса не предусмотрено стандартом языка SQL, но, тем не менее, подавляющее большинство диалектов содержат операторы создания/удаления индексов.

Наличие индекса может позволить многократно ускорить выполнение запроса.

Таблицы SQL Server используют один из двух методов организации страниц данных внутри секции.

Кластеризованные таблицы – это таблицы, имеющие кластеризованный индекс.

Строки данных хранятся по порядку ключа кластеризованного индекса. Кластеризованный индекс реализуется в виде структуры индекса сбалансированного дерева, которая поддерживает быстрый поиск строк по их ключевым значениям в кластеризованном индексе. Страницы в каждом уровне индекса, включая страницы данных на конечном уровне, связаны в двунаправленный список. Однако перемещение из одного уровня на другой выполняется при помощи ключевых значений.

Кучи – это таблицы, которые не имеют кластеризованного индекса. Строки данных хранятся без определенного порядка, и какой-либо порядок в последовательности страниц данных отсутствует. Страницы данных не связаны в связный список.

Индексированные представления имеют такую же структуру хранения, что и кластеризованные таблицы.

Если куча или кластеризованная таблица содержит несколько секций, каждая секция имеет структуру кучи или сбалансированного дерева, содержащую группу строк для указанной секции. Например, если кластеризованная таблица содержит четыре секции, то имеется четыре сбалансированных дерева, по одному на каждую секцию.

Некластеризованные индексы имеют индексную структуру сбалансированного дерева, схожую со структурой кластеризованных индексов. Различие состоит в том, что некластеризованные индексы не влияют на порядок строк данных. Конечный уровень содержит строки индекса. Каждая строка индекса содержит некластеризованное ключевое значение, указатель строки и любые включённые неключевые столбцы. Указатель ссылается на строку данных, которая имеет ключевое значение.

Для каждого столбца **xml** в таблице могут быть созданы один первичный и несколько вторичных XML-индексов. XML-индекс представляет собой разделённое и сохранённое представление больших двоичных объектов XML (BLOB) в столбце типа данных **xml**. XML-индексы хранятся во внутренних таблицах. Для просмотра сведений об XML-индексах можно воспользоваться представлениями каталогов `sys.xml_indexes` и `sys.internal_tables`.

Так как кластеризованный индекс определяет порядок строк (вернее, страниц) в таблице, то у таблицы может быть только один кластеризованный индекс. В качестве ключа кластеризованного индекса нельзя использовать столбцы со следующими типами данных: **varchar(max)**, **nvarchar(max)**, **varbinary(max)** или **xml**.

Так как можно использовать только один кластеризованный индекс для таблицы, необходимо гарантировать, что он будет использоваться для получения максимально возможных преимуществ. Перед созданием кластеризованного индекса необходимо понять, как будет осуществляться доступ к данным. Так как кластеризованный индекс определяет порядок, в котором строки данных таблицы хранятся в SQL Server 2008, кластеризованные индексы больше подходят для конкретных типов данных и шаблонов использования.

Кластеризованные индексы наиболее эффективны при использовании для поддержки запросов, выполняющих следующие действия:

- возвращение диапазона значений с помощью таких операторов, как **BETWEEN**, **>**, **>=**, **<** и **<=**. Поскольку данные таблицы физически хранятся в порядке индекса, то после нахождения с помощью кластеризованного индекса строки с первым значением гарантируется, что последующие индексированные значения физически находятся по соседству;
- возвращение данных, отсортированных с помощью предложения **ORDER BY** или **GROUP BY**. Индекс для столбцов, указанных в предложении **ORDER BY** или **GROUP BY**, может избавить ядро БД от необходимости сортировать данные, так как строки уже отсортированы. При этом производительность запроса повышается;

- возвращение данных, объединённых с помощью предложений JOIN, обычно это столбцы внешних ключей;
- возвращение больших наборов данных.

Если для индексированной таблицы создан кластеризованный индекс, столбец или столбцы, определённые в кластеризованном индексе, автоматически добавляются в конец каждого некластеризованного индекса таблицы. Также к ключу индекса могут быть добавлены столбцы, не входящие непосредственно в ключ индекса. Это может позволить дополнительно увеличить производительность в тех запросах, для которых индекс окажется покрывающим (в индекс входят все поля, используемые в запросе). В случае покрывающего индекса оптимизатор запроса получает все необходимые данные из индекса, минуя обращение непосредственно к таблице. С другой стороны, включение большого числа столбцов в индекс увеличивает физический объём, занимаемый данными, а также ведёт к увеличению уровней индексного дерева. Существуют ограничения для индекса на общее число ключевых столбцов, входящих в составной индекс (их количество не может превышать шестнадцати), и на их общую длину (900 байтов). При этом не учитывается количество и длина присоединённых (неключевых) столбцов.

Для таблицы может быть определён один кластеризованный и до 249 некластеризованных индексов. К выбору и количеству индексов следует подходить с осторожностью. Следует помнить, что индексы увеличивают физический объём, занимаемый БД, а также отрицательно влияют на производительность операций вставки, модификации и удаления данных.

Задача определения оптимального числа индексов для таблицы, а также выбора для них ключевых полей является достаточно сложной и не всегда может быть успешно решена на этапе проектирования БД. К счастью, добавление, изменение и удаление индексов не влияет на схему БД или конструкцию приложений, что позволяет вносить корректировки с состав и ключи индексов без каких-либо опасений. Исключение может составить только создание уникального индекса для таблицы, содержащей повторяющиеся значения в ключевых (для индекса) полях.

Замечание 9. Создание индекса для таблицы, содержащей небольшое количество данных, обычно является неэффективным.

В SQL Server 2008 можно секционировать не только таблицы, но и индексы. Аналогично секционированным таблицам, секционированные индексы являются кластеризованными или некластеризованными индексами, в которых страницы индекса разделены горизонтально на несколько физических местоположений в зависимости от диапазона значений в столбце индекса. Физическим местоположением для секций являются файловые группы. Причина секционирования индексов совпадает с причиной секционирования таблиц: чтобы повысить производительность и управляемость больших индексов, позволяя сфокусировать управление на отдельных секциях, а не на индексе в целом.

Индекс, секционированный подобно его таблице, называется «выровненным» с таблицей. Секционированный индекс является выровненным, если он соответствует следующим требованиям:

- секционирующий ключ совместим с соответствующим ключом таблицы;
- число секций индекса совпадает с числом секций таблицы;
- диапазон значений, хранящихся в секциях, соответствует секциям таблицы.

Доступность свободного пространства в странице индекса может заметно повлиять на производительность операций обновления индекса. Если должна быть вставлена запись индекса, а свободное пространство отсутствует, должна быть создана новая страница индекса, а содержание старой страницы разделено между двумя этими страницами. Такие действия, если происходят слишком часто, могут повлиять на производительность. В SQL Server 2008 существует два параметра, позволяющих управлять объемом свободного пространства, поддерживаемого внутри индекса: FILLFACTOR и PAD_INDEX.

Параметр FILLFACTOR позволяет задать для страниц индекса конечного уровня процент свободного пространства (от 0 до 100). Этот процент определяет, насколько должны быть заполнены страницы конечного уровня. Например, при коэффициенте заполнения 65 процентов заполняется 65 процентов страницы конечного уровня, а 35 процентов пространства страницы свободно для новых строк. По умолчанию SQL Server всегда оставляет достаточно места для размещения хотя бы одной строки индекса максимального размера для каждой страницы неконечного уровня, независимо от значения коэффициента заполнения. Количество элементов страницы неконечного уровня индекса никогда не может быть меньше двух, независимо от значения коэффициента заполнения.

Параметр PAD_INDEX позволяет определить, применяется или нет коэффициент заполнения, применяемый к страницам конечного уровня, и к страницам неконечного уровня. Параметр PAD_INDEX можно использовать, только если определен параметр FILLFACTOR, так как значение процента PAD_INDEX определяется на основе процентного значения, определенного для FILLFACTOR.

Синтаксис оператора создания индекса в MS SQL Server 2008.

```
CREATE [ UNIQUE ] [ CLUSTERED | NONCLUSTERED ] INDEX index_name
ON <object> ( column [ ASC | DESC ] [ ,...n ] )
[ INCLUDE ( column_name [ ,...n ] ) ]
[ WHERE <filter_predicate> ]
[ WITH ( <relational_index_option> [ ,...n ] ) ]
[ ON { partition_scheme_name ( column_name )
    | filegroup_name
    | default
    }
]
[ FILESTREAM_ON { filestream_filegroup_name | partition_scheme_name |
"NULL" } ]
[ ; ]
```

```

<object> ::=
{
  [ database_name. [ schema_name ] . | schema_name. ]
  table_or_view_name
}

```

```

<relational_index_option> ::=
{
  PAD_INDEX = { ON | OFF }
  | FILLFACTOR = fillfactor
  | SORT_IN_TEMPDB = { ON | OFF }
  | IGNORE_DUP_KEY = { ON | OFF }
  | STATISTICS_NORECOMPUTE = { ON | OFF }
  | DROP_EXISTING = { ON | OFF }
  | ONLINE = { ON | OFF }
  | ALLOW_ROW_LOCKS = { ON | OFF }
  | ALLOW_PAGE_LOCKS = { ON | OFF }
  | MAXDOP = max_degree_of_parallelism
  | DATA_COMPRESSION = { NONE | ROW | PAGE }
  [ ON PARTITIONS ( { <partition_number_expression> | <range> }
  [ , ...n ] ) ]
}

```

```

<filter_predicate> ::=
  <conjunct> [ AND <conjunct> ]

```

```

<conjunct> ::=
  <disjunct> | <comparison>

```

```

<disjunct> ::=
  column_name IN (constant ,...)

```

```

<comparison> ::=
  column_name <comparison_op> constant
<comparison_op> ::=
  { IS | IS NOT | = | <> | != | > | >= | != | < | <= | !< }

```

```

<range> ::=
<partition_number_expression> TO <partition_number_expression>

```

Аргументы.

UNIQUE – создаёт уникальный индекс для таблицы или представления. Уникальным является индекс, в котором не может быть двух строк с одним и

тем же значением ключа индекса. Кластеризованный индекс представления должен быть уникальным.

CLUSTERED – создаёт кластеризованный индекс. Если аргумент **CLUSTERED** не указан, создаётся некластеризованный индекс.

NONCLUSTERED – создаёт некластеризованный индекс. Данное значение используется по умолчанию.

index_name – имя индекса. Имена индексов должны быть уникальными в пределах таблицы или представления, но необязательно должны быть уникальными в пределах БД.

column – столбец или столбцы, на которых основан индекс. Имена одного или нескольких столбцов для создания комбинированного индекса. Столбцы, которые должны быть включены в составной индекс, указываются в скобках за аргументом *table_or_view_name* в порядке сортировки.

[**ASC** | **DESC**] – определяет сортировку значений заданного столбца индекса: по возрастанию или по убыванию. Значение по умолчанию — **ASC**.

INCLUDE (*column* [,... *n*]) – указывает неключевые столбцы, добавляемые на конечный уровень некластеризованного индекса. Имена столбцов в списке **INCLUDE** не могут повторяться и не могут использоваться одновременно как ключевые и неключевые.

WHERE <filter_predicate> – создаёт отфильтрованный индекс путём указания строк для включения в индекс. Отфильтрованный индекс должен быть некластеризованным индексом для таблицы. Также создаётся статистика фильтрации для строк данных отфильтрованного индекса.

ON *partition_scheme_name* (*column_name*) – задаёт схему секционирования, которая определяет файловые группы соответствующие секциям секционированного индекса. Схема секционирования должна быть уже создана в БД с помощью инструкции **CREATE PARTITION SCHEME** или **ALTER PARTITION SCHEME**. Аргумент *column_name* задаёт столбец, по которому будет секционирован индекс. Этот столбец должен соответствовать типу данных, длине и точности аргумента функции секционирования, которую использует схема *partition_scheme_name*. Аргумент *column_name* может указывать на столбцы, не входящие в определение индекса. Если аргумент *partition_scheme_name* или *filegroup* не задан и таблица секционирована, индекс помещается в ту же схему секционирования и с тем же столбцом секционирования, что и для базовой таблицы.

ON *filegroup_name* – создаёт заданный индекс в указанной файловой группе. Если местоположение не указано и таблица или представление не секционированы, индекс использует ту же файловую группу, что и базовая таблица или базовое представление. Файловая группа должна существовать.

ON "default" – создаёт заданный индекс в файловой группе, используемой по умолчанию.

[**FILESTREAM_ON** { *filestream_filegroup_name* | *partition_scheme_name* | "NULL" }] – указывает размещение данных **FILESTREAM** для таблицы при создании кластеризованного индекса. Предложение **FILESTREAM_ON** позво-

ляет перемещать данные FILESTREAM в другую файловую группу FILESTREAM или схему секционирования. Аргумент *filestream_filegroup_name* — имя файловой группы FILESTREAM. В файловой группе должен быть определен один файл для файловой группы.

<object>::= — полное или неполное имя индексируемого объекта.

database_name — имя БД.

schema_name — имя схемы, к которой принадлежит таблица или представление.

table_or_view_name — имя индексируемой таблицы или представления.

<relational_index_option>::= — указывает параметры, которые должны использоваться при создании индекса.

PAD_INDEX = { ON | OFF } — определяет заполнение индекса. Значение по умолчанию — OFF.

ON — процент свободного места, определяемый аргументом *fillfactor*, применяется к страницам индекса промежуточного уровня.

OFF или *fillfactor* не указан — страницы промежуточного уровня заполняются почти полностью, при этом остается достаточно места, по крайней мере, для одной строки максимального размера, возможного в этом индексе при заданном наборе ключей на промежуточных страницах.

FILLFACTOR = *fillfactor* — указывает, на сколько процентов должен компонент Database Engine заполнить страницы конечного уровня при создании или перестройке индекса. *fillfactor* должен быть целым числом от 1 до 100. Значение по умолчанию — 0. Если *fillfactor* равен 100 или 0, компонент Database Engine создаёт индексы с полностью заполненными страницами конечного уровня. Аргумент FILLFACTOR действует только при создании или перестройке индекса. Компонент Database Engine не сохраняет динамически указанный процентный объём свободного места на страницах.

SORT_IN_TEMPDB = { ON | OFF } — указывает, сохранять ли временные результаты сортировки в БД **tempdb**. Значение по умолчанию — OFF.

ON — промежуточные результаты сортировки, которые используются при индексировании, хранятся в БД **tempdb**.

OFF — промежуточные результаты сортировки хранятся в той же БД, где и индекс.

IGNORE_DUP_KEY = { ON | OFF } — определяет ответ на ошибку, случающуюся, когда операция вставки пытается вставить в уникальный индекс повторяющиеся значения ключа. Параметр IGNORE_DUP_KEY применяется только к операциям вставки, производимым после создания или перестроения индекса. Значение по умолчанию — OFF.

ON — если в уникальный индекс вставляются повторяющиеся значения ключа, выводится предупреждающее сообщение. С ошибкой завершаются только строки, нарушающие ограничение уникальности.

OFF — если в уникальный индекс вставляются повторяющиеся значения ключа, выводится сообщение об ошибке. Будет выполнен откат всей операции INSERT.

STATISTICS_NORECOMPUTE = { ON | OFF } – определяет функции автоматического обновления статистики для таблицы. Значение по умолчанию – OFF.

ON – устаревшие статистики не пересчитываются автоматически.

OFF – автоматическое обновление статистических данных включено.

DROP_EXISTING = { ON | OFF } – указывает, что названный существующий кластеризованный или некластеризованный индекс удаляется и перестраивается. Значение по умолчанию – OFF.

ON – существующий индекс удаляется и перестраивается. Указанное имя индекса должно совпадать с уже существующим индексом, но определение индекса может быть изменено.

OFF – выдается ошибка, если индекс с указанным именем уже существует.

ONLINE = { ON | OFF } – определяет, будут ли базовые таблицы и связанные индексы доступны для запросов и изменения данных во время операций с индексами. Значение по умолчанию – OFF.

ALLOW_ROW_LOCKS = { ON | OFF } – указывает, разрешена ли блокировка строк. Значение по умолчанию – ON.

ON – блокировки строк допустимы при доступе к индексу. Необходимость в блокировке строк определяет компонент Database Engine.

OFF – блокировки строк не используются.

ALLOW_PAGE_LOCKS = { ON | OFF } – указывает, разрешена ли блокировка страниц. Значение по умолчанию – ON.

ON – блокировки страниц возможны при доступе к индексу. Необходимость в блокировке строк определяет компонент Database Engine.

OFF – блокировки страниц не используются.

MAXDOP = *max_degree_of_parallelism* – переопределяет параметр конфигурации «максимальная степень параллелизма» на время операций с индексами. MAXDOP можно использовать для ограничения числа процессоров, используемых в одновременном выполнении планов. Максимальное число процессоров – 64.

DATA_COMPRESSION – задаёт режим сжатия данных для указанного индекса, номера секции или диапазона секций.

NONE – индекс или заданные секции не сжимаются.

ROW – для индекса или заданных секций производится сжатие строк.

PAGE – для индекса или заданных секций производится сжатие страниц.

ON PARTITIONS ({ <partition_number_expression> | <range> } [,...*n*]) – указывает секции, к которым применяется параметр DATA_COMPRESSION. Если индекс не секционирован, аргумент ON PARTITIONS создаст ошибку. Если не указано предложение ON PARTITIONS, то параметр DATA_COMPRESSION применяется ко всем секциям секционированного индекса.

< partition_number_expression > можно указать одним из следующих способов:

- указав номер секции, например ON PARTITIONS (2);

- указав номера нескольких секций, разделив их запятыми, например ON PARTITIONS (1, 5);
- указав диапазоны секций и отдельные секции, например ON PARTITIONS (2, 4, 6 TO 8).

< range > можно указать номерами секций, разделенными ключевым словом TO, например: ON PARTITIONS (6 TO 8).

Пример 11. Создание индекса с включёнными столбцами.

```
USE AdventureWorks2008;
```

```
GO
```

```
CREATE NONCLUSTERED INDEX IX_Address_PostalCode
```

```
ON Person.Address (PostalCode)
```

```
INCLUDE (AddressLine1, AddressLine2, City, StateProvinceID);
```

```
GO
```

Для столбцов типа **xml** можно создавать XML-индексы. При этом индексируются все теги, значения и пути хранимых в столбце экземпляров XML, и повышается эффективность обработки запросов.

XML-индексы разделяются на следующие категории:

- первичные XML-индексы;
- вторичные XML-индексы.

Первым индексом, создаваемым для столбца типа данных **xml**, должен быть первичный XML-индекс. При наличии первичного XML-индекса поддерживаются вторичные индексы трех типов: PATH, VALUE и PROPERTY. Эти вторичные индексы могут способствовать повышению производительности выполнения разных типов запросов.

Синтаксис оператора создания XML-индекса в MS SQL Server 2008.

```
CREATE [ PRIMARY ] XML INDEX index_name
```

```
ON <object> ( xml_column_name )
```

```
[ USING XML INDEX xml_index_name
```

```
[ FOR { VALUE | PATH | PROPERTY } ] ]
```

```
[ WITH ( <xml_index_option> [ ,...n ] ) ]
```

```
[ ; ]
```

<object> ::=

```
{
  [ database_name. [ schema_name ] . | schema_name. ]
  table_name
}
```

<xml_index_option> ::=

```
{
  PAD_INDEX = { ON | OFF }
  | FILLFACTOR = fillfactor
  | SORT_IN_TEMPDB = { ON | OFF }
```

```

| IGNORE_DUP_KEY = OFF
| STATISTICS_NORECOMPUTE = { ON | OFF }
| DROP_EXISTING = { ON | OFF }
| ONLINE = OFF
| ALLOW_ROW_LOCKS = { ON | OFF }
| ALLOW_PAGE_LOCKS = { ON | OFF }
| MAXDOP = max_degree_of_parallelism
}

```

Аргументы.

[PRIMARY] XML – создаёт XML-индекс по заданному столбцу типа **xml**. Если присутствует ключевое слово PRIMARY, создаётся кластеризованный индекс с ключом, образованным из ключа кластеризации таблицы пользователя и идентификатора XML-узла. Для каждой таблицы можно создать до 249 XML-индексов. Чтобы создать вторичный XML-индекс для столбца типа **xml**, первичный XML-индекс для этого столбца уже должен существовать. XML-индекс может быть создан только для одного столбца типа **xml**.

index_name – имя индекса. Имена индексов должны быть уникальными в пределах таблицы, но не обязательно должны быть уникальными в пределах БД.

xml_column_name – столбец типа **xml**, на котором основан индекс. В одном определении XML-индекса может быть задан только один столбец типа **xml**, но для одного столбца типа **xml** можно создать несколько вторичных XML-индексов.

USING XML INDEX *xml_index_name* – указывает первичный XML-индекс, который должен использоваться при создании вторичного XML-индекса.

FOR { VALUE | PATH | PROPERTY } – указывает тип вторичного XML-индекса.

VALUE – создаёт вторичный XML-индекс для столбцов, где ключевые столбцы (значение узла и путь) входят в первичный XML-индекс.

PATH – создаёт вторичный XML-индекс для столбцов, построенных на основе значений путей и узлов в первичном XML-индексе. Во вторичном индексе типа PATH значениями путей и узлов являются ключевые столбцы, обеспечивающие эффективный поиск путей.

PROPERTY – создаёт вторичный XML-индекс по столбцам первичного XML-индекса (ПК, путь и узел), где ПК — первичный ключ базовой таблицы.

Остальные аргументы совпадают с аналогичными для реляционного индекса.

Пример 12. Создание первичного XML-индекса.

```

USE AdventureWorks2008;
GO
CREATE PRIMARY XML INDEX PXML_ProductModel_CatalogDescription
ON Production.ProductModel (CatalogDescription);
GO

```

Пример 13. Создание вторичного XML-индекса.

USE AdventureWorks2008;

GO

CREATE XML INDEX IXML_ProductModel_CatalogDescription_Path

ON Production.ProductModel (CatalogDescription)

USING XML INDEX PXML_ProductModel_CatalogDescription FOR PATH ;

GO

Задание 1. Создание секционированной таблицы.

Для выполнения данного задания используется БД AdventureWorks2008 и MS SQL Server 2008.

1.1. Создать секционированную таблицу Sales>ReturnsArchiv. Таблица должна содержать следующие поля:

- ReturnID int IDENTITY NOT NULL,
- ProductID int NOT NULL,
- CustomerID int NOT NULL,
- ReturnDate datetime NOT NULL.

Разбиение на секции производится на основании значения года поля ReturnDate. Таблица должна быть заполнена соответствующими значениями из таблиц Sales.SalesOrderHeader (CustomerID, OrderDate) и Sales.SalesOrderDetail (ProductID). Все секции помещаются в одну файловую группу.

Для проверки выполнить запрос, который возвращает строки таблицы Sales>ReturnsArchiv с указанием номера секции, в которую помещена данная строка:

```
SELECT ReturnID, ReturnDate, $Partition.pf_ReturnDate(ReturnDate) PartitionNo
FROM Sales>ReturnsArchive;
```

1.2. Создать БД с необходимым количеством файловых групп. Создать в ней таблицу ReturnsArchiv с данными, полученными в предыдущем задании. Таблица должна быть разбита на секции по тому же принципу, что и в задании 1.1. Каждая секция должна размещаться в своей файловой группе.

1.3. В БД из задания 1.2. создать дополнительную файловую группу, которая зарезервирована под создание новых секций. Изменить при необходимости описание функции и схемы секционирования для увеличения количества секций таблицы с включением в дополнительную секцию данных текущего года.

1.4. Создать в БД из задания 1.2. секционированную таблицу с количеством секций большим, чем количество файловых групп. Файловых групп должно быть не менее трёх. Распределить секции таким образом, чтобы в каждой файловой группе их оказалось не менее одной.

1.5. Создать в БД своего варианта (лабораторная работа № 3) не менее одной секционированной таблицы. Обосновать свой выбор схемы секционирования.

Задание 2. Создание моментального снимка БД.

Для выполнения данного задания используется БД AdventureWorks2008 и MS SQL Server 2008.

2.1. Создать моментальный снимок БД AdventureWorks2008.

2.2. Выполнить запрос, возвращающий из таблицы Person.Person данные о сотруднике с BusinessEntityID=1 с использованием БД AdventureWorks2008 и её моментального снимка.

2.3. Внести изменения в данные об имени сотрудника с BusinessEntityID=1.

2.4. Повторить пункт 2.2. и убедиться в том, что данные в таблице Person.Person БД AdventureWorks2008 изменились, а в моментальном снимке – нет.

2.5. Создать моментальный снимок БД с несколькими файловыми группами, созданной при выполнении задания 1.

Задание 3. Создание представлений.

Для выполнения данного задания используется БД AdventureWorks2008 и MS SQL Server 2008.

3.1. Создать представление, выводящее данные о сотрудниках компании (схема Person). В данные о сотрудниках включить фамилию сотрудника, его имя, адрес, телефон и должность.

3.2. Создать запрос, возвращающий данные о сотрудниках из созданного представления.

3.3. Создать представление аналогично заданию 3.1. В представление включить только сотрудников, проживающих в Сиетле (Seattle).

3.4. Выяснить, является ли представление, созданное в задании 3.2., обновляемым. Если нет, то предложить вариант изменения представления с целью сделать его обновляемым.

3.5. Создать несколько представлений (не менее двух) в БД своего варианта. Обосновать свой выбор. Выяснить, какие из представлений являются обновляемыми, а какие нет. Почему?

Задание 4. Создание и обслуживание индексов.

Для выполнения данного задания используется БД AdventureWorks2008 и MS SQL Server 2008.

4.1. Используя SQL Server Management Studio создать индекс IX_Contact_LastName_FirstName со следующими параметрами:

- тип индекса: некластеризованный;
- ключевые столбцы: LastName и FirstName;
- включённые столбцы: Title, MiddleName и Suffix;
- коэффициент заполнения: 65%;
- блокировка: разрешить оперативную обработку DML-инструкций во время создания индекса.

4.2. Используя Transact-SQL создать индекс Contact_LastName_FirstName_Transact с теми же параметрами, что и индекс IX_Contact_LastName_FirstName.

4.3. Используя Transact-SQL создать для таблицы Production.Product новый уникальный, некластеризованный индекс:

- ключевые столбцы: ProductNumber, Color, ReorderPoint, SafetyStockLevel;
- включённые столбцы: DaysToManufacture;

- блокировка: разрешить блокировку строк, не разрешать блокировку страниц;
- коэффициент заполнения: 90% для конечных и неконечных узлов.

4.4. Создать первичный индекс для xml-столбца, используя следующие настройки:

- имя таблицы: Production.Illustration;
- имя столбца: Diagram;
- имя индекса: PXML_Illustration_Diagram;
- тип индекса: кластеризованный, неуникальный;
- коэффициент заполнения: 60% для конечных и неконечных узлов.

4.5. Создать вторичный индекс для столбца , используя следующие настройки:

- имя индекса: XMLPATH_Illustration_Diagram;
- тип индекса: некластеризованный, неуникальный;
- коэффициент заполнения: 70% для конечных и неконечных узлов.

4.6. Получить сведения обо всех индексах для таблицы Production.Product. Для выполнения задания воспользоваться хранимыми процедурами sp_helpindex и sp_help.

4.7. Изучить представления каталога, обеспечивающие сведения об индексах и тип предоставляемых сведений:

- sys.indexes;
- sys.index_columns;
- sys.stats;
- sys.stats_columns;
- sys.xml_columns.

4.8. Используя помощника по настройке ядра БД (Database Engine Tuning Advisor), провести анализ использования индекса в БД AdventureWorks2008. Для анализа использовать следующие параметры:

- анализируются таблицы Product и SalesOrderDetail;
- анализ ведётся без ограничения по времени;
- использовать индексы;
- не использовать секции;
- сохранять всю существующую структуру БД.

Используется следующая рабочая нагрузка:

```
USE AdventureWorks2008
```

```
SELECT *
```

```
FROM Production.Product
```

```
ORDER BY Name ASC;
```

```
SELECT Name, ProductNumber, ListPrice AS Price
```

```
FROM Production.Product
```

```
WHERE ProductLine = 'R' AND DaysToManufacture < 4
```

```
ORDER BY Name ASC;
```



```

SELECT p.Name AS ProductName,
NonDiscountSales = (OrderQty * UnitPrice),
Discounts = ((OrderQty * UnitPrice) * UnitPriceDiscount)
FROM Production.Product p
    INNER JOIN Sales.SalesOrderDetail sod ON p.ProductID = sod.ProductID
ORDER BY ProductName DESC;

SELECT 'Total income is', ((OrderQty * UnitPrice) * (1.0 - UnitPriceDiscount)),
' for ',p.Name AS ProductName
FROM Production.Product p
    INNER JOIN Sales.SalesOrderDetail sod ON p.ProductID = sod.ProductID
ORDER BY ProductName ASC;

```

Сохранить все рекомендации помощника по настройке ядра БД в виде скрипта.

Экспортировать результаты сеанса помощника по настройке ядра БД в xml-файл.

4.9. Провести анализ степени фрагментации индексов для таблицы Production.Product. Для анализа использовать функцию sys.dm_db_index_physical_stats.

Пример 14. Использование функции sys.dm_db_index_physical_stats.

```

SELECT a.index_id, name, avg_fragmentation_in_percent
FROM sys.dm_db_index_physical_stats (DB_ID(N'AdventureWorks'),
OBJECT_ID(N'Production.Product'), NULL, NULL, NULL) AS a
JOIN sys.indexes AS b ON a.object_id = b.object_id AND a.index_id =
b.index_id;

```

4.10. Выбрать индексы, для которых необходимо проведение дефрагментации. Провести для таких индексов операции реорганизации или перестройки.

4.11. Создать несколько индексов различного типа (кластеризованный, некластеризованный, возможно, XML) для своего варианта. Обосновать свой выбор.

2.8. Лабораторная работа № 7. Создание триггеров и хранимых процедур

Цель работы: создание триггеров и хранимых процедур.

Содержание работы

1. Создание триггеров.

Триггеры являются одной из разновидностей хранимых процедур. Их исполнение происходит при выполнении для таблицы какого-либо оператора DML. Триггеры часто создаются для обеспечения ссылочной целостности или согласованности логически связанных данных, хранящихся в разных таблицах. Пользователи не могут обходить триггеры, поэтому разработчик имеет возможность, используя средства Transact-SQL, реализовать с помощью триггеров сложную бизнес-логику, которую было бы трудно или вообще невозможно реализовать с помощью других механизмов контроля целостности данных.

Триггер – это откомпилированная SQL-процедура, исполнение которой обусловлено наступлением определённых событий внутри реляционной БД. Несмотря на удобство триггеров, их использование часто связано с дополнительными затратами ресурсов на операции ввода/вывода. В том случае, когда тех же результатов можно добиться с помощью хранимых процедур или прикладных программ, применение триггеров нецелесообразно.

Создает триггер только владелец БД. Это ограничение позволяет избежать случайного изменения структуры таблиц, способов связи с ними других объектов и т.п.

В отличие от обычной подпрограммы, триггер выполняется неявно в каждом случае возникновения триггерного события, к тому же он не имеет аргументов. Приведение его в действие иногда называют запуском триггера. С помощью триггеров достигаются следующие цели:

1. проверка корректности введённых данных и выполнение сложных ограничений целостности, которые трудно, если вообще возможно, поддерживать с помощью ограничений целостности, установленных для таблицы;
2. выдача предупреждений, напоминающих о необходимости выполнения некоторых действий при обновлении таблицы, реализованном определённым образом;
3. накопление аудиторской информации посредством фиксации сведений о внесённых изменениях и тех лицах, которые их выполнили;
4. поддержка репликации.

При условии правильного использования триггеры могут стать очень мощным механизмом. Основное их преимущество заключается в том, что стандартные функции сохраняются внутри БД и согласованно активизируются при каждом её обновлении. Это может существенно упростить приложения. При всех своих достоинствах триггеры обладают и недостатками:

- сложность: при перемещении некоторых функций в БД усложняются задачи её проектирования, реализации и администрирования;
- скрытая функциональность: перенос части функций в БД и сохранение их в виде одного или нескольких триггеров иногда приводит к сокрытию от пользователя некоторых функциональных возможностей;
- влияние на производительность: перед выполнением каждой команды по изменению состояния БД СУБД должна проверить триггерное условие с целью выяснения необходимости запуска триггера для этой команды. Выполнение подобных вычислений сказывается на общей производительности СУБД, а в моменты пиковой нагрузки её снижение может стать особенно заметным. Очевидно, что при возрастании количества триггеров увеличиваются и накладные расходы, связанные с такими операциями.

Неправильно написанные триггеры могут привести к серьёзным проблемам, таким, например, как возникновение тупиков. Триггеры способны длительное время блокировать множество ресурсов, поэтому следует обратить особое внимание на сведение к минимуму конфликтов доступа.

В MS SQL Server 2008 определены следующие виды триггеров:

- DML-триггер;
- DDL-триггер;
- Logon-триггер.

Общий формат оператора CREATE TRIGGER:

DML-триггер

```
CREATE TRIGGER [ schema_name. ]trigger_name
ON { table | view }
[ WITH <dml_trigger_option> [ ,...n ] ]
{ FOR | AFTER | INSTEAD OF }
{ [ INSERT ] [ , ] [ UPDATE ] [ , ] [ DELETE ] }
[ WITH APPEND ]
[ NOT FOR REPLICATION ]
AS { sql_statement [ ; ] [ ,...n ] | EXTERNAL NAME <method specifier [ ; ] > }
```

```
<dml_trigger_option> ::=
    [ ENCRYPTION ]
    [ EXECUTE AS Clause ]
```

```
<method_specifier> ::=
    assembly_name.class_name.method_name
```

DDL-триггер

```
CREATE TRIGGER trigger_name
ON { ALL SERVER | DATABASE }
[ WITH <ddl_trigger_option> [ ,...n ] ]
{ FOR | AFTER } { event_type | event_group } [ ,...n ]
AS { sql_statement [ ; ] [ ,...n ] | EXTERNAL NAME < method specifier > [ ; ] }
```

```
<ddl_trigger_option> ::=
    [ ENCRYPTION ]
    [ EXECUTE AS Clause ]
```

```
<method_specifier> ::=
    assembly_name.class_name.method_name
```

Logon-триггер

```
CREATE TRIGGER trigger_name
ON ALL SERVER
[ WITH <logon_trigger_option> [ ,...n ] ]
{ FOR | AFTER } LOGON
AS { sql_statement [ ; ] [ ,...n ] | EXTERNAL NAME < method specifier > [ ; ] }
```

```
<logon_trigger_option> ::=
```

[ENCRYPTION]
[EXECUTE AS Clause]

<method_specifier> ::=
assembly_name.class_name.method_name

Аргументы.

schema_name — имя схемы, которой принадлежит триггер DML. Триггеры DML ограничены областью схемы таблицы или представления, для которых они созданы. Аргумент *schema_name* не может быть указан для триггеров DDL или входа.

trigger_name — имя триггера. Аргумент *trigger_name* должен соответствовать правилам для идентификаторов — за исключением того, что *trigger_name* не может начинаться с символов # или ##.

table | *view* — таблица или представление, в которых выполняется триггер DML, иногда указывается как таблица триггера или представление триггера. Указание уточнённого имени таблицы или представления не является обязательным. На представление может ссылаться только триггер INSTEAD OF. Триггеры DML не могут быть описаны в локальной или глобальной временных таблицах.

DATABASE — применяет область действия триггера DDL к текущей БД. Если этот аргумент определён, триггер срабатывает всякий раз при возникновении в БД события типа *event_type* или *event_group*.

ALL SERVER — применяет область действия триггера DDL или триггера входа к текущему серверу. Если этот аргумент определён, триггер срабатывает всякий раз при возникновении в любом месте на текущем сервере события типа *event_type* или *event_group*.

WITH ENCRYPTION — затемняет текст инструкции CREATE TRIGGER. Использование аргумента WITH ENCRYPTION не позволяет публиковать триггер как часть репликации SQL Server. Параметр WITH ENCRYPTION не может быть указан для триггеров CLR.

EXECUTE AS — указывает контекст безопасности, в котором выполняется триггер. Позволяет управлять учётной записью пользователя, используемой экземпляром SQL Server для проверки разрешений на любые объекты БД, используемые триггером.

FOR | AFTER — тип AFTER указывает, что триггер DML срабатывает только после успешного выполнения всех операций в инструкции SQL, запускаемой триггером. Все каскадные действия и проверки ограничений, на которые имеется ссылка, должны быть успешно завершены, прежде чем триггер сработает. Если единственным заданным ключевым словом является FOR, аргумент AFTER используется по умолчанию. Триггеры AFTER не могут быть определены на представлениях.

INSTEAD OF — указывает, что триггер DML срабатывает вместо инструкции SQL, используемой триггером.

Замечание 1. На каждую инструкцию INSERT, UPDATE или DELETE в таблице или представлении может быть определено не более одного триггера INSTEAD OF. Однако можно определить представления на представлениях, где у каждого представления есть собственный триггер INSTEAD OF. Триггеры INSTEAD OF не разрешены для обновляемых представлений, использующих параметр WITH CHECK OPTION.

{ [DELETE] [,] [INSERT] [,] [UPDATE] } – определяет инструкции изменения данных, по которым срабатывает триггер DML, если он применяется к таблице или представлению. Необходимо указать как минимум одну инструкцию. В определении триггера разрешены любые их сочетания в любом порядке.

Замечание 2. Для триггеров INSTEAD OF параметр DELETE не разрешён в таблицах, имеющих ссылочную связь с указанием каскадного действия ON DELETE. Точно так же параметр UPDATE не разрешён в таблицах, имеющих ссылочную связь с указанием каскадного действия ON UPDATE.

event_type – имя события языка Transact-SQL, которое после выполнения вызывает срабатывание триггера DDL.

event_group – имя стандартной группы событий языка Transact-SQL. Триггер DDL срабатывает после возникновения любого события языка Transact-SQL, принадлежащего к группе *event_group*. После завершения инструкции CREATE TRIGGER параметр *event_group* работает в режиме макроса, добавляя охватываемые им типы события в представление каталога **sys.trigger_events**.

WITH APPEND – указывает, что требуется добавить триггер существующего типа. Аргумент WITH APPEND не может быть использован для триггеров INSTEAD OF или при явном указании триггера AFTER. Аргумент WITH APPEND может использоваться только при указании параметра FOR без INSTEAD OF или AFTER из соображений поддержки обратной совместимости. Аргумент WITH APPEND не может быть указан, если указан параметр EXTERNAL NAME (в случае триггера CLR). Данный аргумент является устаревшим и будет удалён в следующей версии MS SQL Server, что делает его использование в новых проектах крайне нежелательным.

NOT FOR REPLICATION – указывает, что триггер не может быть выполнен, если агент репликации изменяет таблицу, используемую триггером.

sql_statement – условия и действия триггера. Условия триггера указывают дополнительные критерии, определяющие, какие события — DML, DDL или событие входа — вызывают срабатывание триггера.

Замечание 3. Действия триггера, указанные в инструкциях языка Transact-SQL, вступают в силу после попытки использования операции. Триггеры могут содержать любое количество инструкций языка Transact-SQL любого типа, за некоторыми исключениями. Триггеры разработаны для контроля или изменения данных на основании инструкций модификации или определения данных; они не возвращают пользователю никаких данных. Триггеры DML используют логические (концептуальные) таблицы **deleted** и **inserted**. По своей структуре они подобны таблице, на которой определён триггер, то есть таблице, к которой применяется действие пользователя. В таблицах **deleted** и **inserted** содержатся

старые или новые значения строк, которые могут быть изменены действиями пользователя.

`< method_specifier >` – указывает метод сборки для связывания с CLR-триггером. Метод не должен иметь аргументов и возвращать значение типа `void`. Аргумент `class_name` должен быть допустимым идентификатором SQL Server, а в сборке должен существовать класс с таким именем, видимый во всей сборке. Если класс имеет имя, содержащее точки («.») для разделения частей пространства имён, имя класса должно быть заключено в квадратные скобки ([]) или двойные кавычки (" "). Класс не может быть вложенным.

Триггеры DML часто используются для соблюдения бизнес-правил и целостности данных. В SQL Server декларативное ограничение ссылочной целостности обеспечивается инструкциями `ALTER TABLE` и `CREATE TABLE`. Однако декларативное ограничение ссылочной целостности не обеспечивает ссылочную целостность между БД. Ограничение ссылочной целостности подразумевает выполнение правил связи между первичными и внешними ключами таблиц. Если ограничения распространяются на таблицу триггера, они проверяются после срабатывания триггера `INSTEAD OF` и до выполнения триггера `AFTER`. В случае нарушения ограничения выполняется откат действий триггера `INSTEAD OF`, и триггер `AFTER` не срабатывает.

Первые и последние триггеры `AFTER`, которые будут выполнены в таблице, могут быть определены с использованием процедуры `sp_settriggerorder`. Для таблицы можно определить только один первый и один последний триггер для каждой из операций `INSERT`, `UPDATE` и `DELETE`. Если в таблице есть другие триггеры `AFTER`, они будут выполняться случайным образом.

Если инструкция `ALTER TRIGGER` меняет первый или последний триггер, первый или последний набор характеристик изменённого триггера удаляется, а порядок сортировки должен быть установлен заново с помощью процедуры `sp_settriggerorder`.

Триггер `AFTER` выполняется только после того, как вызывающая срабатывание триггера инструкция SQL была успешно выполнена. Успешное выполнение также подразумевает завершение всех ссылочных каскадных действий и проверки ограничений, связанных с изменёнными или удалёнными объектами.

Если триггер `INSTEAD OF`, определённый для таблицы, выполняет по отношению к таблице какую-либо инструкцию, которая бы снова вызвала срабатывание триггера `INSTEAD OF`, триггер рекурсивно не вызывается. Вместо этого инструкция обрабатывается так, как если бы у таблицы отсутствовал триггер `INSTEAD OF`, и начинается применение последовательности ограничений и выполнение триггера `AFTER`.

Если триггер `INSTEAD OF`, определённый для представления, выполняет по отношению к представлению какую-либо инструкцию, которая бы снова вызвала срабатывание триггера `INSTEAD OF`, триггер рекурсивно не вызывается. Вместо этого инструкция выполняет изменение базовых таблиц, на которых основано представление.

Инструкция CREATE TRIGGER должна быть первой инструкцией в пакете и может применяться только к одной таблице.

Триггер создается только в текущей БД, но может, тем не менее, содержать ссылки на объекты за пределами текущей БД.

Если для уточнения триггера указано имя схемы, имя таблицы необходимо уточнить таким же образом.

Одно и то же действие триггера может быть определено более чем для одного действия пользователя (например, INSERT и UPDATE) в одной и той же инструкции CREATE TRIGGER.

Триггеры INSTEAD OF DELETE/UPDATE нельзя определить для таблицы, у которой есть внешний ключ, определенный для каскадного выполнения операции DELETE/UPDATE.

Внутри триггера может быть использована любая инструкция SET. Выбранный параметр SET остается в силе во время выполнения триггера, после чего настройки возвращаются в предыдущее состояние.

Во время срабатывания триггера результаты возвращаются вызывающему приложению так же, как и в случае с хранимыми процедурами. Чтобы предотвратить вызванное срабатыванием триггера возвращение результатов приложению, не следует включать инструкции SELECT, возвращающие результат, или инструкции, которые выполняют в триггере присвоение переменных. Триггер, содержащий либо инструкции SELECT, которые возвращают результаты пользователю, либо инструкции, выполняющие присвоение переменных, требует особого обращения; эти возвращаемые результаты должны быть перезаписаны во все приложения, в которых разрешены изменения таблицы триггера. Если в триггере происходит присвоение переменной, следует использовать инструкцию SET NOCOUNT в начале триггера, чтобы предотвратить возвращение каких-либо результирующих наборов.

SQL Server разрешает рекурсивный вызов триггеров, если с помощью инструкции ALTER DATABASE включена настройка RECURSIVE_TRIGGERS. В рекурсивных триггерах могут возникать косвенная и прямая рекурсии.

Отключение настройки RECURSIVE_TRIGGERS предотвращает выполнение только прямых рекурсий. Чтобы отключить косвенную рекурсию, необходимо с помощью хранимой процедуры **sp_configure** присвоить параметру сервера **nested triggers** значение 0.

Если один из триггеров выполняет инструкцию ROLLBACK TRANSACTION, никакие другие триггеры, вне зависимости от уровня вложенности, не срабатывают.

Вложенность триггеров может достигать максимум 32 уровня. Если триггер изменяет таблицу, для которой определён другой триггер, то запускается второй триггер, вызывающий срабатывание третьего и т.д. Если любой из триггеров в цепочке отключает бесконечный цикл, то уровень вложенности превышает допустимый предел, и срабатывание триггера отменяется. Для отмены вложенных триггеров необходимо присвоить значение 0 параметру **nested triggers** хранимой процедуры **sp_configure**. В конфигурации по умолчанию вложенные

триггеры разрешены. Если вложенные триггеры отключены, рекурсивные триггеры тоже будут отключены, вне зависимости от настройки `RECURSIVE_TRIGGERS`, установленной с помощью инструкции `ALTER DATABASE`.

Пример 1. Создание триггера DML, отправляющего клиенту сообщение, когда кто-то пытается добавить или изменить данные в таблице **Sales.Customer**.

```
USE AdventureWorks2008;
GO
IF OBJECT_ID(N'Sales.uCustomer', N'TR') IS NOT NULL
    DROP TRIGGER Sales.uCustomer;
GO
CREATE TRIGGER Sales.uCustomer
ON Sales.Customer
WITH ENCRYPTION
AFTER INSERT, UPDATE
AS RAISERROR ('Сообщить в головной офис', 16, 10);
GO
```

Пример 2. Изменение триггера из примера 1.

```
USE AdventureWorks2008;
GO
ALTER TRIGGER Sales.uCustomer
ON Sales.Customer
AFTER INSERT
AS RAISERROR ('Сообщить в головной офис ', 16, 10);
GO
```

Пример 3. Триггер обновляет столбец **DiscontinuedDate** в таблице **Production.Products** при каждом удалении какой-либо подкатегории (т. е. при удалении записи из таблицы **Production.ProductSubcategory**). Для всех продуктов, затрагиваемых таким удалением, в поле **DiscontinuedDate** записывается текущее значение даты в знак того, что они изымаются из продажи.

```
CREATE TRIGGER Production.uDelSubcategory ON Production.ProductSubcategory
AFTER DELETE AS
BEGIN
UPDATE P SET [DiscontinuedDate] = SYSDATETIMEOFFSET()
FROM Production.Products P INNER JOIN deleted as d
ON P.CategoryID = d.CategoryID
END;
```

Пример 4. Создание триггера `INSTEAD OF`.

```
CREATE TRIGGER [delEmployee] ON [HumanResources].[Employee]
INSTEAD OF DELETE NOT FOR REPLICATION AS
BEGIN
    SET NOCOUNT ON;
```



```

DECLARE @DeleteCount int;
SELECT @DeleteCount = COUNT(*) FROM deleted;
IF @DeleteCount > 0
BEGIN
    RAISERROR
    (N'Employees cannot be deleted. They can only be marked as not
    current.', -- Message
    10, -- Severity.
    1); -- State.
    -- Roll back any active or uncommittable transactions
    IF @@TRANCOUNT > 0
    BEGIN
        ROLLBACK TRANSACTION;
    END
END;
END;

```

Задание 1. Создание триггеров.

1.1. Создать триггер DELETE (INSTEAD OF), который будет копировать сведения о кандидатах (таблица **HumanResources.JobCandidate**) в новую таблицу **HumanResources.JobCandidateHistory** (необходимо создать самостоятельно) в случае, если кто-либо удалит данные о них из таблицы **HumanResources.JobCandidate**. Необходимо также записывать в поле **RejectedDate** текущую дату с помощью функции **GETDATE**.

1.2. Создать триггер DELETE (INSTEAD OF) для таблицы **Sales.SalesOrderHeader**, который будет блокировать удаление записи при наличии ссылающихся на неё строк из таблицы **Sales.SalesOrderDetail**.

1.3. Создать триггер INSERT (INSTEAD OF), который будет разрешать создавать новую строку в таблице **Sales.SalesOrderDetail** только в том случае, если количество заказанного товара (**OrderQty**) не превосходит складского запаса (таблица **Production.Product**, поле **ReorderPoint**).

1.4. Создать триггер INSERT (AFTER), который будет уменьшать количество продукта на складе (таблица **Production.Product**, поле **ReorderPoint**) при создании новой строки в таблице **Sales.SalesOrderDetail**.

1.5. Создать необходимые триггеры UPDATE, которые будут разрешать изменение данных в таблице **Sales.SalesOrderDetail** только в том случае, если новое запрошенное количество может быть предоставлено заказчику. При этом следует внести необходимые изменения в данные на складе.

1.6. Создать необходимые триггеры UPDATE, INSERT, DELETE, которые будут разрешать изменение данных в таблице **Sales.SalesOrderDetail** только в том случае, когда заказ находится в стадии выполнения (таблица **Sales.SalesOrderHeader**, поле **Status=1**).

1.7. Создать несколько триггеров для БД своего варианта. Созданные триггеры должны представлять все типы DML-триггеров (INSTEAD OF, AFTER для INSERT, UPDATE и DELETE).

2. Создание хранимых процедур.

Задание 2. Создание хранимых процедур.

2.1. Создать хранимую процедуру, реализующую следующий запрос:

```
SELECT Name, ProductNumber, DaysToManufacture
FROM Production.Product
WHERE DaysToManufacture >= 1
ORDER BY DaysToManufacture DESC, Name
```

Выполнить созданную процедуру.

2.2. Изменить хранимую процедуру (задание 2.1) таким образом, чтобы она принимала в качестве параметра количество дней, требуемых для производства продукции (поле **DaysToManufacture**). В теле процедуры установить проверку на корректность значения параметра.

2.3. Создать хранимую процедуру **GetDiscounts** в схеме **Sales**, которая извлекает следующие столбцы из таблицы **Sales.SpecialOffer**: **Description**, **DiscountPct**, **Type**, **Category**, **StartDate**, **EndDate**, **MinQty** и **MaxQty**. Процедура должна возвращать все строки, отсортированные по параметрам **StartDate** и **EndDate**.

2.4. Создать хранимую процедуру **GetDiscountsForCategory** в схеме **Sales**, которая принимает входной параметр **@Category**, имеющий тип данных **nvarchar** и принимающий до 50 символов. Эта процедура должна извлекать те же столбцы, что и для **GetDiscounts**, но должна фильтровать строки на основе параметра **@Category**.

2.5. Создать хранимую процедуру **GetDiscountsForCategoryAndDate** в схеме **Sales**, которая принимает параметр **@Category**, как и **GetDiscountsForCategory**, но включает дополнительный входной параметр **@DateToCheck datetime**. Параметр **@DateToCheck** должен иметь возможность принимать стандартное значение NULL. Если значение NULL указано для параметра **@DateToCheck**, задать для этого параметра значение текущих даты и времени, используя функцию **GETDATE**. Эта процедура должна извлекать те же столбцы, что и для **GetDiscounts**, но должна фильтровать строки на основе параметров **@Category** и **@DateToCheck**.

2.6. Создать хранимую процедуру, реализующую получение сведений (название, цена) о продукте, продаваемом в указанный пользователем период (поля **SellStartDate** и **SellEndDate**). В теле процедуры предусмотреть проверку на корректность значений параметров.

2.7. Создать процедуру **AddDiscount** в схеме **Sales**, которая вставляет новые записи в таблицу **Sales.SpecialOffer**. Инstrukция INSERT должна быть защищена соответствующей обработкой ошибок, и любые ошибки должны быть записаны в таблицу **dbo.ErrorLog**. Если новая вставка завершится успешно, параметр **@NewProductID** должен быть обновлен на значение функции

SCOPE_IDENTITY. Возвращаемое значение должно также указывать успех или неудачу вставки.

2.8. Создать хранимую процедуру (набор хранимых процедур), реализующую создание нового заказа на продажу. Количество заказанных позиций (различных продуктов) не превышает пяти. Хранимая процедуры (процедуры) должна создавать новую запись в таблице **Sales.SalesOrderHeader** и необходимое количество новых строк в таблице **Sales.SalesOrderDetail**.

2.9. Создать несколько хранимых процедур для БД своего варианта.

2.9. Лабораторная работа № 8. Создание отчётов с использованием Reporting Services

Цель работы: знакомство с базовыми возможностями Reporting Services.

Содержание работы

1. Обзор основных возможностей Microsoft SQL Server 2008 Reporting Services.

Microsoft SQL Server 2008 Reporting Services предоставляет полную серверную платформу, разработанную для решения широкого спектра задач, связанных с отчётностью, включающую управляемую отчётность по предприятию, отчёты по требованию, встроенные отчёты и веб-отчёты. Службы Reporting Services 2008 содержат инструменты и функции, необходимые для составления отчётов в разнообразных удобных форматах, используя широкий спектр источников данных (ИД).

Службы Reporting Services включают в себя три средства конструирования отчётов:

- конструктор отчётов в среде Business Intelligence Development Studio (BIDS);
- построитель отчётов версии 1.0;
- построитель отчётов версии 2.0.

Конструктор отчётов – это графическое средство для создания полнофункциональных отчётов служб Reporting Services. Он позволяет обращаться ко многим различным типам ИД и создавать детально настроенные отчёты. После создания отчёта предоставляется доступ ко всем функциям управления отчётами служб Reporting Services. Для использования конструктора отчётов необходимо знать, как подключаться к ИД и создавать запросы, но не обязательно знать язык определения отчётов.

Построитель отчётов версии 1.0 – клиентское приложение, помогающее создавать нерегламентированные отчёты. Для использования построителя отчётов версии 1.0 необходимо спроектировать и опубликовать модель отчёта для ИД с помощью проекта модели отчёта в среде BIDS. В построителе отчётов версии 1.0 можно подключить модель отчёта как ИД. С помощью упрощенного интерфейса можно перетащить поля данных в шаблоны отчёта, а затем группировать, сортировать, форматировать данные или создавать подытоги. Можно

пройти по данным в модели отчёта, чтобы увидеть автоматически сформированные отчёты. Отчёты можно экспортировать в другие приложения, такие как Microsoft Office Excel, публиковать или сохранять отчёты локально. Для использования построителя отчётов версии 1.0 не требуется навыков работы с SQL Server.

Построитель отчётов версии 2.0 – интуитивно понятная оптимизированная для работы с Office среда создания отчётов. Построитель отчётов 2.0 предназначен для работы с данными, определения макетов, предварительного просмотра отчёта, а также для публикации отчёта на сервере отчётов или на узле SharePoint. Это приложение включает мастер для создания таблиц и диаграмм, построители запросов и редактор выражений. Он также поддерживает расширенные функции работы с отчётами, доступные в службах SQL Server 2008 Reporting Services.

Отчёт можно также создать программно.

2. Создание простого табличного отчёта.

Чтобы создать отчет в SQL Server, необходимо сначала создать проект сервера отчётов, в котором будет сохранен файл определения отчёта (с расширением RDL) и другие файлы ресурсов, необходимые для отчёта. Затем будет создан действительный файл определения отчёта, определён ИД для него, НД и макет отчёта. При выполнении отчёта происходит получение и объединение фактических данных с макетом, затем осуществляется его подготовка к просмотру на экране, после чего может быть выполнен импорт, печать или сохранение.

Для начала работы требуется запустить BIDS и создать новый проект. Тип проекта – Business Intelligence Projects, шаблон – Report Server Project.

Для создания файла определения отчёта необходимо в окне обозревателя решений создать новый элемент в «Reports». Шаблон этого элемента – Report.

Конструктор отчётов является компонентом служб Reporting Services, запускаемым в среде BIDS. Данные определяются в области **Report Data**. Макет отчёта определяется в представлении **Design**. Отчёт можно выполнить и посмотреть в представлении **Preview**.

Для добавления к отчёту ИД требуется в области **Report Data** выбрать новый ИД. В данном примере требуется имя – **AdventureWorks**, параметр «Embedded cnnnection» выбран, тип соединения – **MS SQL Server**, а строка соединения – **Data source=localhost; initial catalog=AdventureWorks2008** (рисунок 9.1).

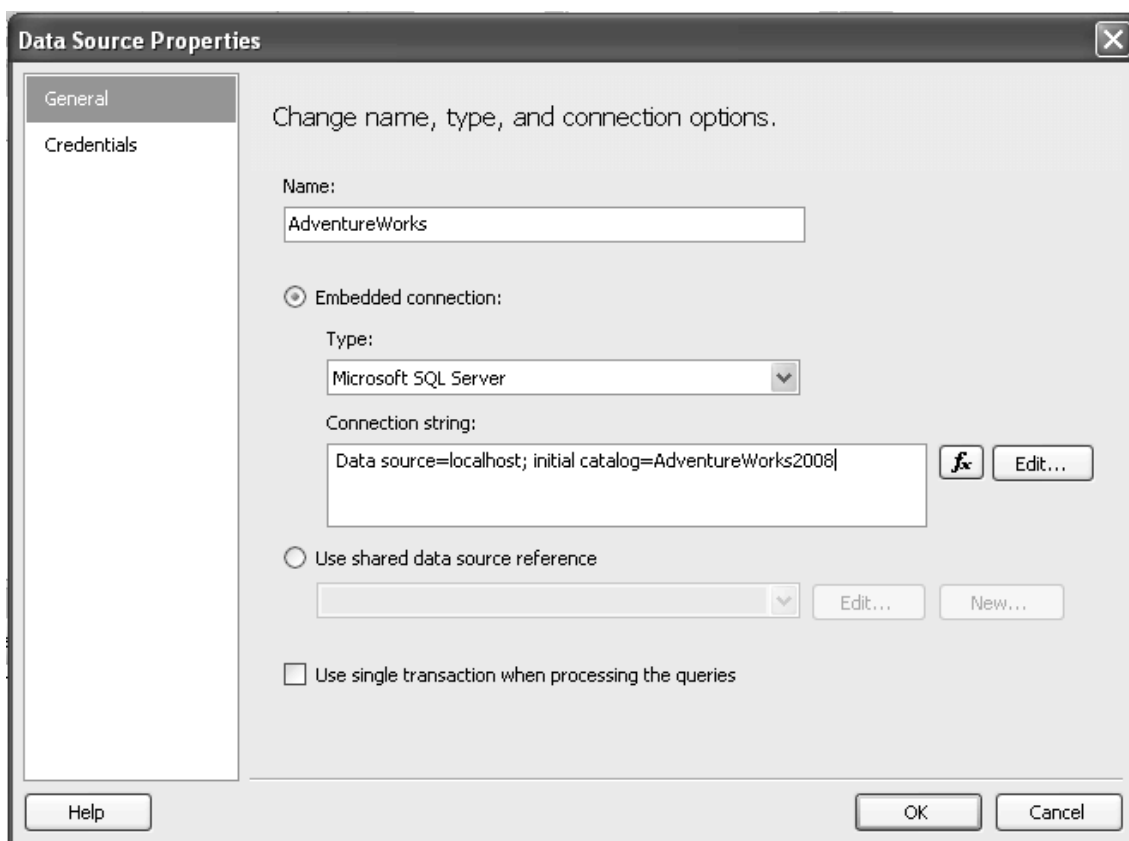


Рисунок 9.1. Свойства источника данных

Замечание 1. Общий вид строки соединения в случае использования версии SQL Server Express with Advanced Services или именованного экземпляра включает сведения об экземпляре: `Data source=localhost\SQLEXPRESS; initial catalog=AdventureWorks2008`.

В области Report Data должен появиться ИД с именем AdventureWorks.

После определения ИД требуется определить НД, который содержит указатель на ИД и запрос, используемый в отчёте, а также вычисляемые поля и переменные. В рассматриваемом примере используется запрос, получающий из БД AdventureWorks2008 сведения о заказах на продажу (рисунок 9.2):

SELECT

soh.OrderDate AS [Date],
soh.SalesOrderNumber AS [Order],
pps.Name AS Subcat, pp.Name as Product,
SUM(sd.OrderQty) AS Qty,
SUM(sd.LineTotal) AS LineTotal

FROM Sales.SalesPerson sp INNER JOIN Sales.SalesOrderHeader AS soh
ON sp.BusinessEntityID = soh.SalesPersonID

INNER JOIN Sales.SalesOrderDetail AS sd ON sd.SalesOrderID =
soh.SalesOrderID

INNER JOIN Production.Product AS pp ON sd.ProductID = pp.ProductID

INNER JOIN Production.ProductSubcategory AS pps

ON pp.ProductSubcategoryID = pps.ProductSubcategoryID

```

INNER JOIN Production.ProductCategory AS ppc
  ON ppc.ProductCategoryID = pps.ProductCategoryID
GROUP BY ppc.Name, soh.OrderDate, soh.SalesOrderNumber,
  pps.Name, pp.Name, soh.SalesPersonID
HAVING ppc.Name = 'Clothing'

```

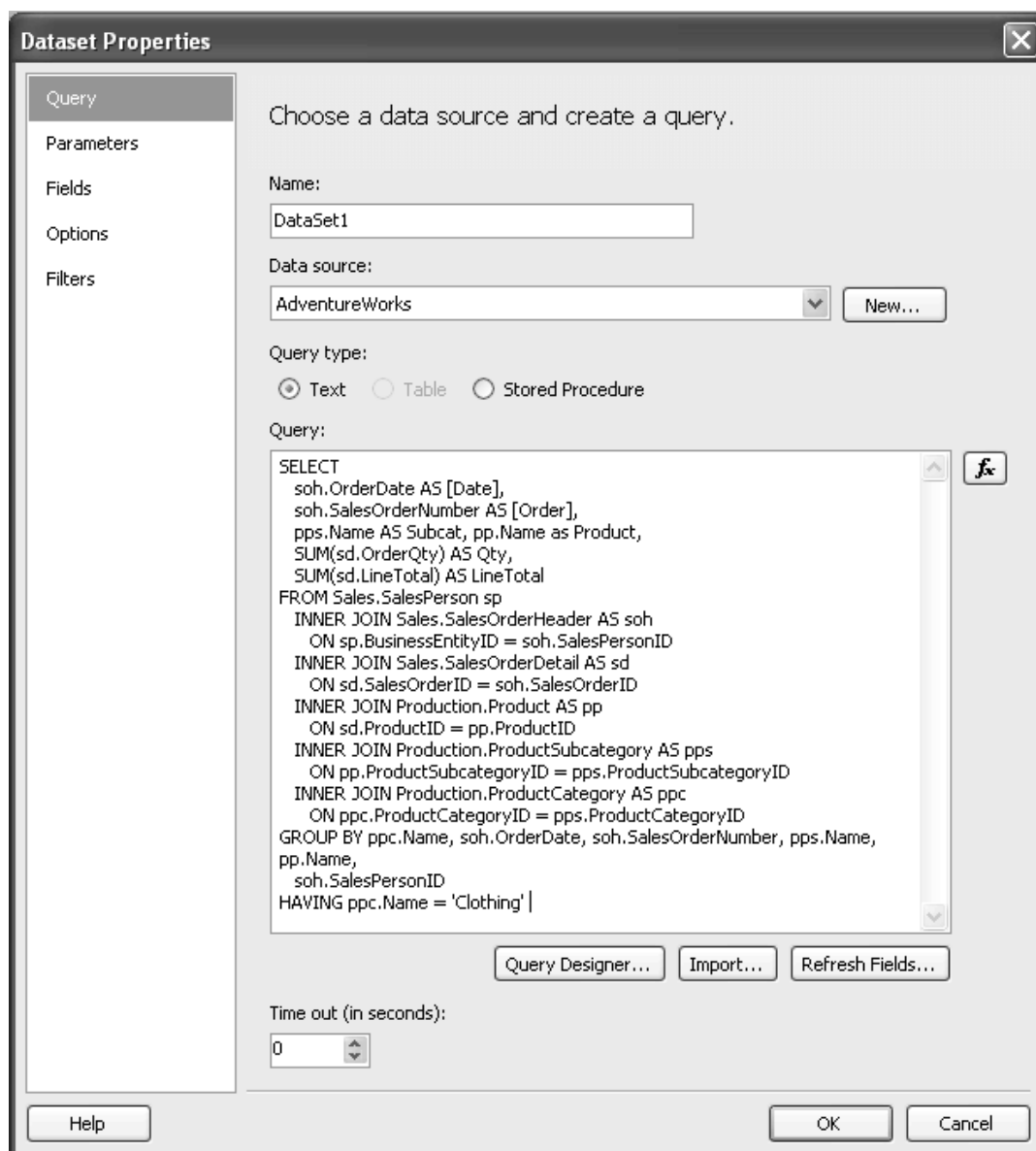


Рисунок 9.2. Создание набора данных

Можно воспользоваться режимом «Query Designer...», позволяющем отобразить запрос, как в текстовом, так и графическом режиме (рисунок 9.3).

Следующим шагом после определения НД является определение макета отчёта. Макет отчёта создаётся путём перетаскивания в область конструктора областей данных, текстовых полей, изображений и других элементов, включаемых в состав создаваемого отчёта. Включение в отчёт таблицы происходит при добавлении элемента «Table» из контекстного меню области конструктора.

Замечание 2. По умолчанию таблица включает три столбца.

	Date	Order	Product
≡	[Date]	[Order]	[Product]

Рисунок 9.4. Таблица с тремя полями.

Для включения в отчёт следующих столбцов, требуется перетаскивать поля в правый край последнего столбца до возникновения вертикального курсора и появления на указателе мыши знака «плюс» [+]. Для окончательного формирования столбцов отчёта требуется включить ещё поля «Qty» и «LineTotal».

Замечание 3. Аналогичным способом можно вставлять столбцы в середину таблицы, перетаскивая нужное поле в правый край того столбца, после которого необходимо поместить новый.

Замечание 4. Для предварительного просмотра подготовленного отчёта достаточно перейти на вкладку «Preview».

Для форматирования какого-либо поля отчёта достаточно перейти на вкладку «Design» и выбрать из контекстного меню соответствующего столбца пункт «Text Box properties» и в свойстве «Number» указать требуемый формат отображения данных.

Изменение стиля заголовка производится достаточно стандартным способом. Например, для выделения заголовка курсивом, можно выделить всю строку и в меню «Формат» выбрать необходимое свойство текста.

Для группировки данных в отчёте необходимо перетащить нужное поле в область **Row groups**, поместив это поле над строкой с обозначением **Details**.

Рассматриваемый отчёт будет включать группировку по полям «Date» и «Order». Исходные столбцы «Date» и «Order», находящиеся справа от двойной линии, следует удалить (рисунок 9.5).

Замечание 5. Вновь появившиеся столбцы необходимо форматировать заново (при необходимости).

Для добавления итогов в отчёт необходимо в контекстном меню нужного столбца выбрать пункт «Add total».

В данном отчёте будут представлены итоги по заказу, дню и общий итог. Для этого в ячейках таблицы «Line Total» и «Qty» необходимо выбрать пункт «Add total». Для наглядности в строку, содержащую итоги можно добавить слово «Итого по заказу» (рисунок 9.6).

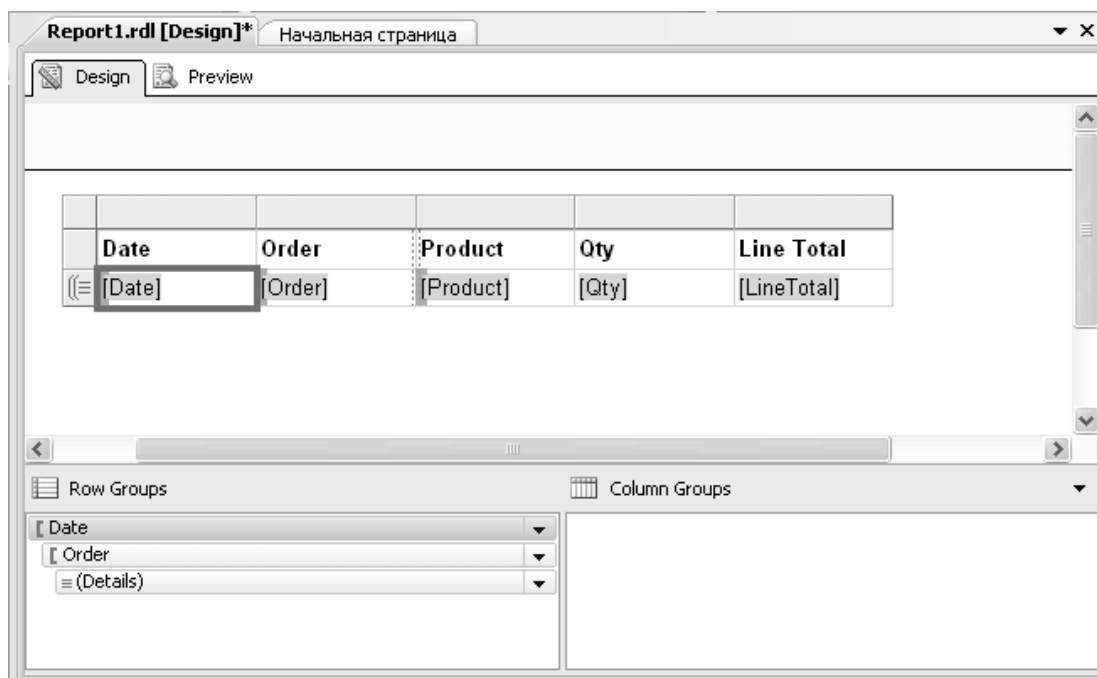


Рисунок 9.5. Макет отчёта с группировками по двум полям.

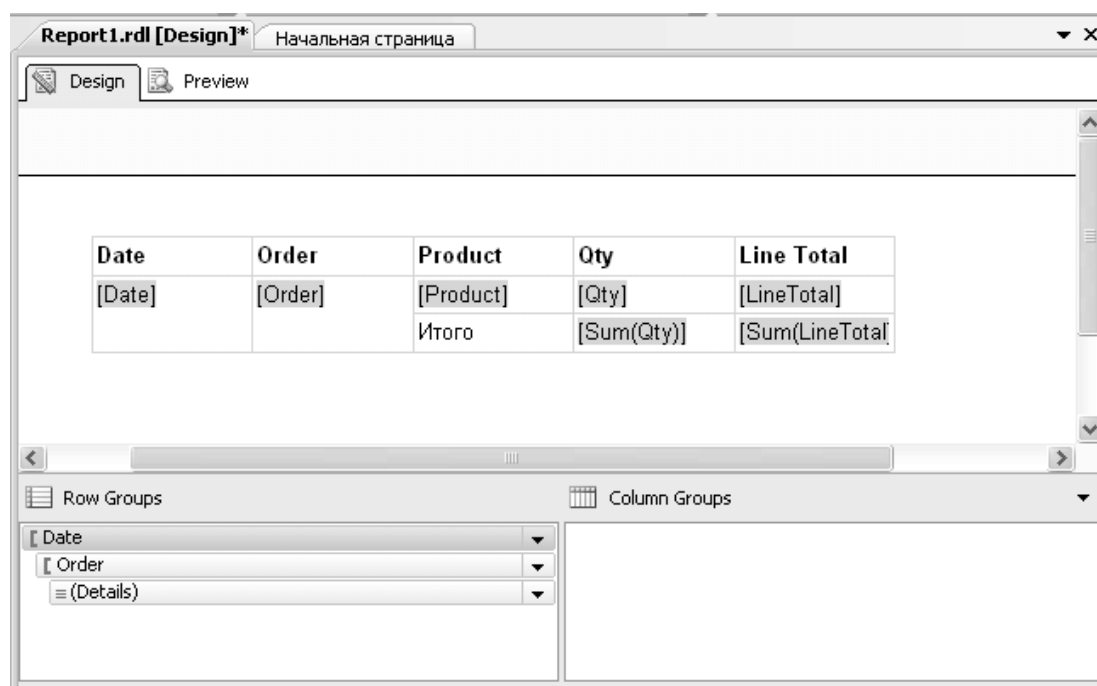


Рисунок 9.6 Добавление итогов по заказу.

Добавление ежедневного отчёта происходит через выбор свойства «Add total» и варианта «After» ячейки с «Order», а итоги по отчёту требуют аналогичной настройки поля «Date» (рисунок 9.7).

	Date	Order	Product	Qty	Line Total
	[Date]	[Order]	[Product]	[Qty]	[LineTotal]
			Итого	[Sum(Qty)]	[Sum(LineTotal)]
		Итого за день		[Sum(Qty)]	[Sum(LineTotal)]
	ИТОГО			[Sum(Qty)]	[Sum(LineTotal)]

Рисунок 9.7. Отчёт с итогами.

3. Добавление параметров в отчёт.

Достаточно часто требуется включить в отчёт только те данные, которые относятся к определённому периоду времени. Для решения этой и других аналогичных задач можно использовать включение параметров в запрос.

Для преобразования ИД в общий ИД необходимо в области «Report Data» выбрать из контекстного меню AdventureWorks команду «Convert to Shared Data Source» («Преобразовать в общий источник данных»). Далее требуется изменить свойство ИД. Для включения параметров в запрос можно в конструкторе запросов изменить текст следующим образом:

```
SELECT
    soh.OrderDate AS [Date],
    soh.SalesOrderNumber AS [Order],
    pps.Name AS Subcat, pp.Name as Product,
    SUM(sd.OrderQty) AS Qty,
    SUM(sd.LineTotal) AS LineTotal
FROM Sales.SalesPerson sp
    INNER JOIN Sales.SalesOrderHeader AS soh
        ON sp.BusinessEntityID = soh.SalesPersonID
    INNER JOIN Sales.SalesOrderDetail AS sd
        ON sd.SalesOrderID = soh.SalesOrderID
    INNER JOIN Production.Product AS pp
        ON sd.ProductID = pp.ProductID
    INNER JOIN Production.ProductSubcategory AS pps
        ON pp.ProductSubcategoryID = pps.ProductSubcategoryID
    INNER JOIN Production.ProductCategory AS ppc
        ON ppc.ProductCategoryID = pps.ProductCategoryID
GROUP BY ppc.Name, soh.OrderDate, soh.SalesOrderNumber,
    pps.Name, pp.Name, soh.SalesPersonID
HAVING (ppc.Name = 'Clothing'
    AND (soh.OrderDate BETWEEN (@StartDate) AND (@EndDate)))
```

По отношению к предыдущему запросу добавлена только одна строка – самая последняя.

Можно задать предварительный просмотр отчёта и убедиться, что происходит запрос двух параметров. По умолчанию @StartDate и @EndDate будут иметь тип данных «Текст». Тип данных можно изменить на требуемый (в дан-

ном случае, Дата-время), выбрав в области «Report Data» папку «Parameters». Там же можно добавить для параметров значение по умолчанию.

При работе с датами бывает удобно оперировать днями недели. Для включения значения дня недели в отчёт можно изменить в конструкторе существующий запрос, добавив новое вычисляемое поле:

DATENAME(weekday,soh.OrderDate) AS Weekday

Наличие этого поля позволяет задать новый параметр отчёта. Для этого необходимо в области «Report Data» создать новый параметр с именем **DayoftheWeek**. В текстовом поле **Prompt** можно задать заголовок параметра (например, День недели для фильтрации). В качестве значения по умолчанию можно выбрать **Friday** (пятница).

Для использования нового параметра можно задать фильтр. Добавление фильтра происходит следующим образом:

- при нажатии правой кнопки мыши маркера строки или столбца появляется контекстное меню, содержащее пункт **Tablix Properties...**;
- на вкладке **Filters** выбирается вариант **Add** (добавить);
- из раскрывающегося списка в поле **Expression** (выражение) выбирается **[Weekday]**;
- параметр оператора имеет значение «Равно»;
- **Value** (значение) устанавливается равным **[@DayoftheWeek]**.

После выполнения перечисленных действий фильтр таблицы настроен на сравнение значения в поле **Weekday** со значением параметра **DayoftheWeek**. В этом легко убедиться, сформировав отчёт.

Доступные (или допустимые) значения представляют список возможных значений параметра отчёта. Допустимые значения можно предоставить из запроса, специально разработанного для получения набора значений из ИД; также можно предоставить стандартный набор значений. С помощью привязки набора доступных значений к запросу ИД, который выполняется при обработке отчёта, можно гарантировать, что из раскрывающегося списка могут быть выбраны только те значения, которые существуют в БД.

Добавление параметров для создания списка допустимых значений рассматривается на примере отбора только данных по продажам, произведённым конкретным менеджером. Для включения в параметры отчёта сведений о менеджере необходимо изменить с помощью конструктора существующий запрос на новый, который отличается от предыдущего наличием дополнительного условия в конструкции HAVING:

soh.SalesPersonID = (@BusinessPersonID)

Для заполнения списка допустимых значений для параметра отчёта необходимо создать новый ИД с именем **BusinessPersons**. В область запроса следует поместить код:

```
SELECT sp.BusinessEntityID, c.FirstName, c.LastName
FROM
    Sales.SalesPerson AS sp
```

INNER JOIN

HumanResources.Employee AS e

ON e.BusinessEntityID = sp.BusinessEntityID

INNER JOIN

Person.Person AS c

ON c.BusinessEntityID = e.BusinessEntityID

ORDER BY sp.BusinessEntityID

В области «Report Data» в НД **BusinessPersons** необходимо добавить вычисляемое поле (в последнее текстовое поле) с именем **Name** и источником `Fields!LastName.Value & ", " & Fields!LastName.Value`. В области «Report Data» необходимо настроить свойства параметра **BusinessPersonID**. В качестве типа данных выбирается Integer, а для допустимых значений указывается вариант `Get values from a query` (Получать значения из запроса), НД – **BusinessPersons**, Value field (Значение) – `BusinessEntityID`, Label field (Метка) – **Name**. Аналогично настраивается значение параметра по умолчанию.

Кроме использования списка допустимых значений можно применять многозначные параметры отчёта, которые позволяют выбирать несколько значений из списка. В качестве таких многозначных параметров будут использоваться **BusinessPersonID** и **DayoftheWeek**.

Так как использование многозначного параметра требует условия на включение кортежа в отчёт в виде принадлежности к списку, необходимо внести изменения в существующий НД. Теперь запрос должен выглядеть следующим образом:

SELECT

soh.OrderDate AS [Date],

DATENAME(weekday, soh.OrderDate) as Weekday,

soh.SalesOrderNumber AS [Order],

pps.Name AS Subcat, pp.Name as Product,

SUM(sd.OrderQty) AS Qty,

SUM(sd.LineTotal) AS LineTotal

FROM Sales.SalesPerson sp

INNER JOIN Sales.SalesOrderHeader AS soh

ON sp.BusinessEntityID = soh.SalesPersonID

INNER JOIN Sales.SalesOrderDetail AS sd

ON sd.SalesOrderID = soh.SalesOrderID

INNER JOIN Production.Product AS pp

ON sd.ProductID = pp.ProductID

INNER JOIN Production.ProductSubcategory AS pps

ON pp.ProductSubcategoryID = pps.ProductSubcategoryID

INNER JOIN Production.ProductCategory AS ppc

ON ppc.ProductCategoryID = pps.ProductCategoryID

GROUP BY ppc.Name, soh.OrderDate, soh.SalesOrderNumber,

pps.Name, pp.Name, soh.SalesPersonID

HAVING ppc.Name = 'Clothing'

```
AND (soh.OrderDate BETWEEN (@StartDate) AND (@EndDate))
AND soh.SalesPersonID IN (@BusinessPersonID)
```

По отношению к предыдущему здесь изменена только последняя строка.

Для изменения параметра отчёта **BusinessPersonID** с однозначного на многозначный, достаточно в его свойствах установить флажок **Allow multiple values**.

Создание многозначного параметра, отвечающего за день недели, требуется добавить НД. В свойствах НД задаётся имя (например, **WeekDaysfromQuery**) и запрос. Для рассматриваемого случая текст запроса выглядит следующим образом:

```
SET DATEFIRST 1;
SELECT DISTINCT
    DATEPART(weekday, s.OrderDate) AS WeekDayNumber,
    DATENAME(weekday, s.OrderDate) AS Weekday
FROM Sales.SalesOrderHeader s
ORDER BY WeekDayNumber
```

Для использования созданного НД в отчёте необходимо изменить свойства параметра **DayoftheWeek**. Кроме возможности использовать данные из запроса требуется установить возможность выбирать несколько значений. Если сейчас попробовать запустить просмотр отчёта, то выйдет сообщение об ошибке. Для восстановления работоспособности отчёта требуется заменить оператор равенства (=) в фильтре (таблица отчёта → Tablix Properties...) на оператор **IN**.

При создании отчётов можно использовать каскадирование параметров. При каскадировании параметров список значений для одного параметра зависит от значения, выбранного для предыдущего параметра. Для каскадных параметров важен порядок следования.

Для создания нового отчёта с использованием каскадных параметров необходимо в обозревателе решений создать новый элемент. Шаблон для создаваемого элемента - **Report**, в качестве имени нового отчёта можно выбрать **CascadingParameters.rdl**.

В отчёте будет использоваться ИД с именем **AdventureWorks_Ref** с установкой **Use shared data source reference** (использовать ссылку на общий источник данных). В качестве ИД необходимо выбрать **AdventureWorks**.

Для нового НД с именем **SalesbyCategory** необходимо задать следующий запрос:

```
SELECT
    pc.Name AS Category,
    psc.Name AS Subcategory,
    p.Name AS Product,
    soh.[OrderDate],
    soh.SalesOrderNumber,
    sd.OrderQty,
    sd.LineTotal
FROM [Sales].[SalesPerson] sp
```

```

INNER JOIN [Sales].[SalesOrderHeader] soh
ON sp.[BusinessEntityID] = soh.[SalesPersonID]
INNER JOIN Sales.SalesOrderDetail sd
ON sd.SalesOrderID = soh.SalesOrderID
INNER JOIN Production.Product p
ON sd.ProductID = p.ProductID
INNER JOIN Production.ProductSubcategory psc
ON p.ProductSubcategoryID = psc.ProductSubcategoryID
INNER JOIN Production.ProductCategory pc
ON pc.ProductCategoryID = psc.ProductCategoryID
WHERE (pc.Name = (@Category)
AND psc.Name = (@Subcategory)
AND p.Name = (@Product))

```

Этот запрос содержит три параметра: @Category, @Subcategory и @Product.

В области «Report Data» для источника данных **AdventureWorks_Ref** требуется создать новый НД (с именем CategoryValues) и запросом

```
SELECT DISTINCT Name AS Category FROM Production.ProductCategory
```

Для параметра @Category необходимо установить в качестве допустимых значения из запроса с использованием НД CategoryValues. В качестве значения для полей Value field и Label fields выбирается единственное поле из запроса – Category.

Параметр @Subcategory будет настраиваться таким образом, чтобы он зависел от предыдущего - @Category. Для этого создаётся новый НД с именем SubcategoryValues и запросом

```

SELECT DISTINCT psc.Name AS Subcategory
FROM Production.ProductSubcategory AS psc
INNER JOIN Production.ProductCategory AS pc
ON pc.ProductCategoryID = psc.ProductCategoryID
WHERE pc.Name = (@Category)

```

Допустимые значения для параметра @Subcategory настраиваются аналогично предыдущему случаю.

Настройка последнего параметра ведётся аналогично двум предыдущим. Новый НД будет иметь имя ProductValues и запрос

```

SELECT DISTINCT p.Name AS Product
FROM Production.Product p
INNER JOIN Production.ProductSubcategory AS psc
ON p.ProductSubcategoryID = psc.ProductSubcategoryID
INNER JOIN Production.ProductCategory AS pc
ON pc.ProductCategoryID = psc.ProductCategoryID
WHERE (pc.Name = (@Category)
AND psc.Name = (@Subcategory))

```

Следующим шагом будет добавление таблицы для отображения результатов. Для этого необходимо в представлении Design добавить таблицу и из НД SalesbyCategory перетащить в неё три поля: SalesOrderNumber, OrderQty, Line-

Total. Из НД SalesbyCategory в область RowGroups необходимо перетащить поля Category, Subcategory, Product и OrderDate, размещая их один под другим выше группы Details.

Замечание 6. При необходимости можно отформатировать ячейки LineTotal (формат Currency) и OrderDate (формат Date).

Для того, чтобы убедиться в работоспособности отчёта, достаточно выполнить его предварительный просмотр. Можно обратить внимание, что по мере выбора каждого последующего параметра в раскрывающемся списке для следующего параметра отображаются только основанные на сделанном выборе допустимые значения.

Таким образом, был создан отчёт, в котором отображаются заказы на продажу, содержащие определенный продукт; в отчёте используются каскадные параметры, которые фильтруют продукт по категории, подкатегории и названию продукта.

Кроме простого табличного можно формировать детализированный отчёт (ДО). ДО – это отчёт, открываемый переходом по ссылке в текущем отчёте. ДО открывается по щелчку на текстовом поле с созданным действием детализации. Если ДО имеет параметры, то необходимо передать значения параметров каждому параметру отчёта.

Начальные действия при создании ДО аналогичны предыдущим. Имя для нового отчёта будет выбираться исходя из его назначения. В рассматриваемом примере используется имя **SalesOrderDetail.rdl**. ИД будет иметь имя **AdventureWorks_Ref** с установкой **Use shared data source reference AdventureWorks**. Новый НД с именем SalesDetails строится с использованием запроса

```
SELECT p.Name AS Product, sd.OrderQty AS Quantity, sd.LineTotal
FROM Sales.SalesOrderDetail AS sd
    INNER JOIN Production.Product AS p
        ON sd.ProductID = p.ProductID
    INNER JOIN Sales.SalesOrderHeader AS soh
        ON sd.SalesOrderID = soh.SalesOrderID
WHERE (soh.SalesOrderNumber = (@SalesOrder) )
ORDER BY sd.SalesOrderDetailID
```

Таблица отчёта будет содержать значения полей Product, Quantity и LineTotal. При желании можно отформатировать таблицу. Для текущей проверки работоспособности отчёта можно запустить его предварительный просмотр со значением параметра, равным SO43659. Пока отчёт отображает линейные итоги указанного товара.

Для включения механизма детализации требуется обратиться к уже созданному отчёту с именем **SalesOrders.rdl**. Следует изменить свойство поля Order. Для этого из контекстного меню выбирается пункт **Text Box Properties**, далее в **Action** для свойства **Enable as a hyperlink** устанавливается значение **Go to report**. В раскрывающемся списке в области **Select a report from the list** выбирается созданный отчёт **SalesOrderDetail**. В области **Use these parameters to**

run the report необходимо добавить **SalesOrder** в качестве имени и **[Order]** в качестве значения. Это действие привяжет значение из основного отчёта к параметру, который ожидает целевой отчёт. Осталось только изменить стиль и цвет текста ссылки детализации. Это выполняется в разделе **Font** свойства текстового поля.

4. Создание простого матричного отчёта.

Первоначальные шаги по созданию матричного отчёта полностью повторяют действия, рассмотренные в предыдущих разделах. Требуется создать новый отчёт с именем **SalesbyAreaandYear.rdl**, новым ИД (имя **AdventureWorksMatrixData**), использующим БД **AdventureWorks2008**. Значения типа **Embedded connection** (внедрённое соединение) и **Connection string** (строка соединения) также аналогичны предыдущим. Новый ИД (имя Sales) определяется с использованием следующего запроса:

```
SELECT
    soh.SalesPersonID AS id,
    p.FirstName,
    p.LastName,
    soh.SalesOrderNumber AS [Order],
    soh.OrderDate AS [Date],
    DATEPART(yy, soh.OrderDate) AS [Year],
    DATEPART(mm, soh.OrderDate) AS [Month],
    st.[Group] AS [Geography],
    st.CountryRegionCode AS CountryRegion,
    st.Name AS Territory,
    ppc.Name AS Category,
    pps.Name AS Subcat,
    pp.Name AS Product,
    pp.Color, PP.Size,
    CASE
        WHEN pp.Size = 'S' THEN 1
        WHEN pp.Size = 'M' THEN 2
        WHEN pp.Size = 'L' THEN 3
        WHEN pp.Size = 'XL' THEN 4
    ELSE pp.Size
    END AS SizeSortOrder,
    SUM(sd.OrderQty) AS Qty,
    SUM(sd.LineTotal) AS LineTotal
FROM
    Sales.SalesPerson AS sp
    INNER JOIN Sales.SalesOrderHeader AS soh
        ON sp.BusinessEntityID = soh.SalesPersonID
    INNER JOIN Person.Person AS p
        ON p.BusinessEntityID = sp.BusinessEntityID
```



```

INNER JOIN Sales.SalesOrderDetail AS sd
    ON sd.SalesOrderID = soh.SalesOrderID
INNER JOIN Production.Product AS pp
    ON sd.ProductID = pp.ProductID
INNER JOIN Sales.SalesTerritory AS st
    ON st.TerritoryID = sp.TerritoryID
INNER JOIN Production.ProductSubcategory AS pps
    ON pp.ProductSubcategoryID = pps.ProductSubcategoryID
INNER JOIN Production.ProductCategory AS ppc
    ON ppc.ProductCategoryID = pps.ProductCategoryID
GROUP BY ppc.Name, soh.OrderDate, soh.SalesOrderNumber, pps.Name,
    pp.Name, soh.SalesPersonID, p.LastName, p.FirstName,
    st.[Group], st.CountryRegionCode, st.Name, pp.Color, pp.Size
HAVING (DATEPART(yy,soh.OrderDate) IN ('2003','2004')
    AND st.[Group] = 'North America'
    AND LEFT(pps.Name,1) IN ('C','T')
    AND LEFT(ppc.Name,1) = 'C')

```

В результирующем наборе отображаются данные из 18 полей семи разных таблиц БД **AdventureWorks2008**. В этом отчёте есть разные поля, которые можно использовать для группирования данных, включая год и месяц даты заказа, географическое расположение территории продажи (страна и область), категория и подкатегория продукта. Кроме того, данные о продажах отфильтрованы так, чтобы получить только заказы на продажу за 2003 и 2004 года на территории Северной Америки по категориям **Clothing** и **Components** и по подкатегориям, начинающимся с буквы **C**.

Для создания макета отчёта используется элемент **Matrix** (матрица). В ячейку **Rows** (строки) следует поместить поле **Category**. После этого в ячейке в квадратных скобках отобразится имя поля, оно же отобразится и в заголовке столбца, внутри маркера строки рядом с ячейкой отобразится квадратная скобка (сигнализирует о связи строки с группой), в области группирования отобразится группа строк **Category**. В ячейку **Columns** (столбцы) следует поместить поле **Geography** (в маркере столбца отобразится квадратная скобка, а область группы столбцов будет содержать группу **Geography**). Наконец, в ячейку **Data** (данные отчёта) помещается поле **LineTotal**. Поле **LineTotal** определяется с помощью агрегирующей функции **SUM**, поэтому в ячейку будет отображаться выражение **[Sum(LineTotal)]**.

Для добавления вложенной группы строк следует поместить поле **Subcat** в область **Row Groups** (группы строк), а для добавления вложенной группы столбцов поле **CountryRegion** помещается в область **Columns Groups** (группы столбцов). При этом в таблицу отчёта будут автоматически добавлены новый столбец **Subcat** и новая строка **CountryRegion**.

Для добавления смежной группы столбцов в области **Columns Groups** следует вызвать контекстное меню для группы **Geography**, выбрать **Add Group** → **Adjacent After** (добавить группу → прилегающая после). В диалоговом окне

Tablix Group следует выбрать группировку по году (**Year**). В ячейку в столбце **Year**, смежную с ячейкой [Sum(LineTotal)] необходимо скопировать её значение, что добавит в неё статистическое выражение [Sum(LineTotal)].

При необходимости можно отформатировать матрицу отчёта, выбрав соответствующий стиль отображения для числовых полей, а также заголовков строк и столбцов.

Для добавления итогов строк следует для столбца [Year] выполнить операцию **Add Total** → **After**.

Для некоторых заголовков отчёта желательно выполнить операцию объединения ячеек. В частности, для столбца с надписью [Year] это можно сделать, щёлкнув и перетаскив мышью заголовок вниз на одну ячейку. В результате окажется выделено две ячейки по вертикали. Далее из контекстного меню выбирается команда **Merge Cells** (объединить ячейки). Ячейки могут быть отформатированы с использованием различных вариантов выравнивания, настройки шрифта, стиля и цвета, задания нового имени для заголовка.

5. Дополнительные возможности службы Reporting Services.

В службе Reporting Services существует возможность опубликовать подготовленный отчёт для доступа к нему со стороны других пользователей. В настоящей работе такая возможность не рассматривается.

Ещё одной возможностью является добавление в отчёт диаграммы. Для демонстрации такой возможности будет создан новый отчёт. В качестве ИД будет использоваться AdventureWorks2008. Для ИД используется следующий запрос:

```
SELECT [FullName] AS Name
      ,[2002] AS SalesBeforeLast
      ,[2003] AS SalesLastYear
      ,[2004] AS SalesYTD
FROM [Sales].[vSalesPersonSalesByFiscalYears]
```

Далее в меню **View** (вид) выбирается **Toolbox** → **Chart**. Из предложенных вариантов графического представления результатов отчёта можно выбрать диаграмму любого типа. Для дальнейшего построения отчёта поле «Name» переносится из области Report Data в область добавления полей категорий, а поле «SalesYTD» в область добавления полей данных. Самый простой вариант диаграммы готов.

Для отображения процентов в каждом срезе круговой диаграммы необходимо из контекстного меню диаграммы выбрать **Show Data Labels** (отобразить метки данных). Для любой из появившихся меток выбирается пункт **Series Label properties** (свойства метки ряда) и выбирается вариант **#PERCENT** в поле **Label data** (данные метки). Для указания количества знаков используется вариант **#PERCENT{Pn}**, где n соответствует необходимому числу десятичных разрядов. Для добавления эффекта рисования к круговой диаграмме выбирается пункт меню **View** → **Property Pane** (окно свойств) и для диаграммы в узле **CustomAttributes** параметру **PieDrawingStyle** присваивается значение **SoftEdge**. Для объединения срезов круговой диаграммы в узле

CustomAttributes свойству **PieDrawingStyle** присваивается значение **SingleSlice**, **CollectedThresholdUsePercent** - значение **True**, а **CollectedThreshold** - значение, например, 8.

3. Курсовые работы

3.1. Темы курсовых работ

- 1.1. Учёт работы автотранспортного предприятия.
- 1.2. Учёт работы автостоянки.
- 1.3. Интернет-магазин.
- 1.4. Учёт выдачи и возврата банковских кредитов.
- 1.5. Электронный каталог книг.
- 1.6. Бухгалтерский учёт движения материалов.
- 1.7. Электронный документооборот.
- 1.8. Домашняя бухгалтерия.
- 1.9. Железнодорожная касса.
- 1.10. Продажа авиабилетов.
- 1.11. Начисление и учёт заработной платы.
- 1.12. Учёт работы книжного магазина.
- 1.13. Учёт работы музыкального магазина.
- 1.14. Учёт работы производственного участка.
- 1.15. Складской учёт материалов.
- 1.16. Калькуляция блюд.
- 1.17. Управление гостиницей.
- 1.18. Учёт услуг телефонной станции.
- 1.19. Учёт кадров.
- 1.20. Учёт основных средств.
- 1.21. Домашняя фонотека.
- 1.22. Электронная библиотека.
- 1.23. Кулинарная книга.
- 1.24. Электронная историческая энциклопедия.
- 1.25. Учёт работы зоопарка по содержанию животных.
- 1.26. Учёт работы кадрового агентства.
- 1.27. Учёт работы детского сада.
- 1.28. Учёт работы салона красоты.
- 1.29. Учёт работы таможенного склада.

3.2. Требования к курсовым работам

Обязательными элементами курсовой работы являются:

1. титульный лист;
2. содержание;
3. введение;
4. основная часть;

5. заключение;
6. список литературы;

Дополнительными элементами курсовой работы являются:

1. вспомогательные указатели;
2. приложения.

Обязательные элементы

1. **Титульный лист** является первой страницей курсовой работы и должен содержать следующие сведения: наименование учреждения (учебного заведения), название (тему), сведения о выполнившей курсовую работу, сведения о руководителе, наименование места и год выполнения.
2. **Содержание** включает перечень основных элементов курсовой работы с указанием номеров страниц, с которых начинается их месторасположение.
3. **Введение** характеризует актуальность и социальную значимость рассматриваемой темы, состояние её разработанности в мировой теории и практике, цель и задачи курсовой работы, обоснование выбора используемых методов, особенности курсовой работы и основное смысловое содержание её разделов.

Вводная часть содержит краткий обзор литературы по теме, который должен показать знакомство студента со специальной литературой, его умение систематизировать источники, выделять существенное, оценивать ранее сделанное другими исследователями, определять главное в современном состоянии изученности темы. Обзор работ следует делать только по вопросам выбранной темы, а не по всей проблеме в целом. В обзор включается только та литература, с которой студент ознакомился (знаком) лично.

После формулировки цели предпринимаемого исследования указываются конкретные задачи, которые предстоит решать в соответствии с этой целью. Это обычно делается в форме перечисления (изучить, описать, установить, выявить, вывести формулу, разработать и т.п.). Формулируя задачи, следует учитывать, что описание их решения должно составить содержание глав курсовой работы.

В конце вводной части раскрывается структура работы.

4. **Основная часть** должна включать в себя следующие разделы:

4.1. Проектирование БД:

- инфологическое проектирование БД;
- логическое проектирование БД;
- описание структуры (схема) БД и структура таблиц.

4.2. Реализация БД:

- выбор инструментальных средств СУБД;
- описание проекта; структура программы и данных.

4.3. Интерфейс с пользователем:

- общие спецификации пользовательских функций;
- описание основных запросов и алгоритмов обработки данных.

5. В **заключении** раскрывается значимость рассмотренных вопросов для теории и практики; приводятся выводы, характеризующие итоги проделанной работы, предложения и рекомендации.
6. **Список литературы** – это упорядоченный в алфавитно-хронологической последовательности перечень библиографических описаний документальных источников информации по теме курсовой работы. В списке следует указывать автора, наименование источника, издательство, год издания.

Дополнительные элементы

7. В состав **вспомогательных указателей** могут входить:
 - список сокращений (оформляется в виде алфавитного перечня принятых в курсовой работе сокращений и соответствующих им полных обозначений понятий);
 - список условных обозначений (оформляется в виде перечня используемых в тексте курсовой работы условных обозначений с соответствующей расшифровкой);
 - указатель таблиц и иллюстраций (оформляется в виде перечня названий таблиц или иллюстраций, упорядоченных в соответствии с их порядковыми номерами, с указанием страниц их месторасположения в тексте курсовой работы).
8. **Приложения помещаются** в конце курсовой работы. Каждое приложение должно начинаться с новой страницы и иметь содержательный заголовок. Приложения нумеруются арабскими цифрами по порядковой нумерации. Номер приложения размещается в правом верхнем углу над заголовком приложения после слова «Приложение», после цифры точку не ставится. Приложения должны иметь общую с остальной частью курсовой работы нумерацию страниц. На все приложения в основной части курсовой работы должны быть ссылки.

Созданная в ходе работы над курсовой БД должна содержать данные в объёме, достаточном для демонстрации возможностей созданного приложения и объектов БД.

Список использованной литературы

1. Аткинсон, Леон. MySQL. Библиотека профессионала.: Пер. с англ. — М.: Издательский дом «Вильямс», 2002. — 624 с.: ил. — Парал. тит. англ.
2. Веллинг, Люк. MySQL. Краткое изложение основ работы с MySQL[Текст] : учеб. пособие / Л. Веллинг, Л. Томсон. - М. : Вильямс, 2005. - 304 с.
3. Голицына, Ольга Леонидовна. Системы управления базами данных[Текст] : учеб. пособие / О. Л. Голицына, т. Л. Партыка, И. И. Попов. - М. : ФОРУМ : ИНФРА-М, 2006. - 432 с. : ил. - (Профессиональное образование). - Библиогр.: с. 315-318
4. Кузин, Александр Владимирович. Базы данных[Текст] : учеб. пособие / А. В. Кузин, С. В. Левонисова. - М. : Академия, 2005. - 320 с. - (Высшее про-

- фессиональное образование. Информатика и вычислительная техника). - Библиогр.: с. 313
5. Кузин, Александр Владимирович. Разработка баз данных в системе Microsoft Access[Текст] : учебник / А. В. Кузин, В. М. Демин. - М. : ФОРУМ : ИНФРА-М, 2005. - 224 с. : ил. - Библиогр.: с. 220
 6. Малыхина, Мария Петровна. Базы данных: основы, проектирование, использование[Текст] : учеб. пособие / М. П. Малыхина. - 2-е изд. - СПб. : БХВ-Петербург, 2006. - 528 с. : ил. - Библиогр.: с. 509-513
 7. Фёдоров, Алексей. Microsoft SQL Server 2008. Обзор ключевых новинок. – М.: Издательство «Русская редакция», 2008. – 129 с.
 8. Хомоненко, Анатолий Дмитриевич. Базы данных[Текст] : учебник / А. Д. Хомоненко, В. М. Цыганков, М. Г. Мальцев. - 6-е изд. - М. : Бином-Пресс : КОРОНА-Век, 2007. - 736 с. - Библиогр. в конце глав
 9. Microsoft SQL Server 2005. Реализация и обслуживание [Текст] : учебный курс. - М. : Русская редакция ; СПб. : Питер, 2008. - 742 с. : ил. + 1 эл. опт. диск (CD-ROM). - (Учебный курс Microsoft)
 10. Электронная документация по SQL Server 2008 [Электронный ресурс] / Microsoft. – Электрон. дан. – 2009 – Режим доступа: [http://technet.microsoft.com/ru-ru/library/bb418432\(SQL.10\).aspx](http://technet.microsoft.com/ru-ru/library/bb418432(SQL.10).aspx) свободный. – Загл. с экрана. – Яз. рус.