Алексей Федоров

Microsoft® SQL Server™ 2008

Краткий обзор ключевых новинок

Алексей Федоров

Microsoft® SQL Server™ 2008

Краткий обзор ключевых новинок

Специальное издание для конференции, посвященной запуску
Windows Server 2008, SQL Server 2008, Visual Studio 2008

Оглавление

История Microsoft SQL Server насчитывает уже второй десяток лет, и новый продукт — Microsoft SQL Server 2008 — относится к четвертому поколению продуктов для управления базами данных. Развитие продукта отражало и отражает тенденции, присутствующие в IT-индустрии — если первые версии SQL Server предназначались для простой обработки данных (Load, Query, Insert, Delete), то последующие привнесли в продукт многомерные модели, аналитику, поддержку XML, Data Mining и ряд других функций.

Microsoft SQL Server. Первые десять лет

Началом истории Microsoft SQL Server по праву можно считать 1986 год, когда Microsoft и Sybase выпустили совместную версию продукта — SQL Server 1.0 и адаптировали ее 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 базы данных.

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 система управления базами данных, которая могла конкурировать с аналогичными продуктами на платформе UNIX. В 1994 году Microsoft и Sybase прервали совместное пятилетнее соглашение и бывшие партнеры 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 продуктам 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 систему управления базами данных 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 базы данных с поддержкой масштабируемости, новый процессор обработки запросов, систему самонастройки, самоуправления, а также реализовать поддержку OLAP и ETL с привлечением специалистов из компании Panorama. Разработка новой СУБД заняла около трех 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, распределенные 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 отчетов, трансформации данных, расширенные функции анализа и т. п.

Тенденции

К ключевым факторам, определяющим развитие СУБД, относятся следующие: появление новых типов данных, новых форм-факторов для устройств хранения данных, новые стандарты представления и обмена данными, а также существенное снижение стоимости хранения самих данных.

Ключевые факторы, определяющие развитие СУБД

Наверное самым значительным фактором является объем данных, производимых во всем мире. Если посмотреть на данные, хранимые в электронном виде, то к 1999 году мире насчитывалось 12 Эбайт (10^{18} байт) информации. К середине 2002 года было создано еще столько же. При этом, 92% этой информации хранится на жестких дисках. Интересна еще одна цифра — все страницы Интернета занимают порядка 21 Тбайт, тогда как объем информации, пересылаемой по электронной почте, перекрывает объем интернет-страниц больше чем в 500 раз.

Но настоящим «взрывом» в производстве цифровых данных стали не Интернет и не электронные документы, а различные цифровые устройства — телефоны, фотоаппараты, видеокамеры, проигрыватели и т. п. 55%

всех цифровых данных хранится на компьютерах пользователей и только 16% — в корпоративных хранилищах данных. Более того, к 2009 году в США все телевидение будет цифровым, что должно существенно увеличить объем хранимых в электронном виде данных.

Это означает, что мы храним все больше и больше информации, в структурированном и неструктурированном виде.

И еще один малоизвестный факт — различные датчики поставляют значительные объемы информации. Так, исследование Земли выражается на 2007 год в 15 петабайтах данных, медицинские данные о пациентах приносят еще 1 Эбайт в год, видео-мониторинг — 100 Эбайт в год, двигатели самолетов — 30 петабайт в год и т. д.

Стремительный рост способов создания различных типов цифровой информации сопровождается таким же стремительным падением стоимости ее хранения. В 1980 году гигабайтный диск IBM 3380 имел размер холодильника, весил около 250 кг и стоил 40 000 долл. К 2000 году стоимость хранения 1 Гбайт уже составляла 6.9 долл. , а в 2006 — 0.38 долл. Те же тенденции заметны и в снижении стоимости оперативной памяти — 1.52 долл. за Мбайт в 2000 году и 0.08 дол. в 2006 году. Посмотрим на объемы поставок носителей данных — 145 Пбайт (10^{15} байт) в 2002 году, 422 Пбайт в 2004 и 704 Пбайт в 2006 году. Цена на flash-носители также стремительно падает и в скором времени мы увидим, как такие носители начнут вытеснять жесткие диски.

Что все перечисленные выше факторы означают для индустрии в целом и для производителей СУБД в частности?

- Во-первых, нужно быть готовым к тому, что массивные объемы данных будут храниться в центрах обработки данных, на жестких дисках и на переносных устройствах;
- Типы данных будут как структурированными, так и неструктурированными;
- Платформа обработки данных должна быть масштабируемой, поддерживать все возможные устройства хранения данных, обладать интегрированными средствами управления, обеспечивать высокую производительность и поддерживать исследование неструктурированных данных и мощные средства анализа бизнес-данных.

В процессе своего развития приложения все больше и больше включают поддержку различных типов данных — XML, электронной почты, календарей, файлов, документов, гео-данных и т. п. в дополнение к традиционным типам данных, поддерживаемым на уровне баз данных и файловой системы. Для обеспечения работоспособности таких приложений нужна соответствующая платформа, которая позволяла бы безопасно хранить такие данные, выполнять по ним поиск, анализировать их, синхронизировать и т. д. Как мы увидим далее, ключевые новинки, появившиеся в SQL Server 2008, как раз адресованы для решения описанных выше задач, а также для улучшения работы с «традиционными» данными.

Введение

Ключевые новинки, появившиеся в SQL Server 2008, относятся к четырем основным областям эффективной обработки данных: масштабируемая и управляемая платформа для корпоративных данных, динамичная разработка приложений, работающих с базами данных, расширенные типы данных и способы их хранения и интуитивное представление для приложений класса Business Intelligence.

В SQL Server 2008 реализованы такие возможности, как «прозрачное» шифрование данных, системный аудит и управление внешними ключами, обеспечивающими защиту и безопасность данных, расширенные средства управления серверами и ресурсами — Resorce Governor и Data Compression, а также улучшенные средства анализа и Business Intelligence, появившиеся в SQL Server Integration Services, SQL Server Analysis Services и SQL Server Reporting Services.

Ниже мы рассмотрим ключевые новинки в каждой группе более подробно. Начнем с обсуждения того, как ряд функций SQL Server 2008 позволяет ему служить масштабируемой и управляемой платформой для сбора, обработки и хранения корпоративных данных.

Варианты поставки SQL Server 2008

Планируется выпуск SQL Server 2008 в двух основных редакциях — Standard Edition и Enterprise Edition, а также в редакции, предназначенном для разработчиков — SQL Server 2008 Developer Edition. Последнее представляет собой версию Enterprise Edition, лицензия на которую позволяет разрабатывать и тестировать системы, но использовать ее как платформу для коммерческих продуктов нельзя.

Ключевые характеристики двух основных редакций SQL Server 2008 показаны в следующей таблице.

Ключевые характеристики	Standard Edition	Enterprise Edition
Число процессоров	4	Макс. поддерживаемое операционной системой
Масштабируемость и производительность	Ограниченная	Разделение на партиции (таблицы и индексы)
		Параллельные индексные операции
Высокая доступность	Ограниченная	16-узловая кластеризация
		Полное зеркалирование базы
		Онлайновые операции
		Управление распределением ресурсов
		Сжатие резервных копий
		Добавление процессоров и памяти «на лету»
Корпоративная безопасность	Ограниченная	Поддержка сторонних систем управления ключами
		Прозрачное шифрование данных
		Аудит безопасности
Data Warehouse	—	Масштабируемость для Data-marts и систем генерации отчетов
		Сжатие данных
		Оптимизация запросов
		Change Data Capture (CDC)
Расширенные средства Business Intelligence	—	Расширенные функции аналитики Расширенные алгоритмы для Data-marts
		Подписка на отчеты, управляемые данными
Управляемость на уровне предприятия	—	Автоматическое управление группами серверов

Более подробную информацию о редакциях SQL Server 2008 можно получить на сайте Microsoft по адресу: *http://www.microsoft.com/sql*.

Аппаратные и программные требования

Ниже мы рассмотрим минимальные аппаратные и программные требования для SQL Server 2008.

32-битные версии SQL Server 2008

В следующей таблице показаны системные требования для 32-битной версии **SQL Server 2008 Developer Edition**.

Компонент	Требование
Процессор	Тип процессора: ■ Совместимый с Pentium III Скорость процессора: ■ Минимум: 600 МГц, рекомендуется 1ГГц и быстрее
.NET Framework	При установке SQL Server устанавливаются следующие компоненты, требуемые для обеспечения функционирования продукта: ■ Microsoft .NET Framework 2.0 ■ Microsoft SQL Server Native Client ■ Файлы, требующиеся Microsoft SQL Server Setup
Операционная система	Windows Vista Ultimate Edition Windows Vista Home Premium Edition Windows Vista Home Basic Edition Windows Vista Enterprise Edition Windows Vista Business Edition Windows Vista 64-bit x64 Ultimate Edition Windows Vista 64-bit x64 Home Premium Edition Windows Vista 64-bit x64 Home Basic Edition Windows Vista 64-bit x64 Enterprise Edition Windows Vista 64-bit x64 Business Edition Windows Server 2008 Standard Edition Windows Server 2008 Data Center Edition Windows Server 2008 Enterprise Edition Windows Server 2008 64-bit x64 Standard Edition Windows Server 2008 64-bit x64 Data Center Edition Windows Server 2008 64-bit x64 Enterprise Edition Windows Server 2003 SP1 Windows Server 2003 Enterprise Edition SP1 Windows XP Professional SP2 Windows Server 2003 64-Bit x64 SP1 Windows Server 2003 64-Bit x64 Enterprise Edition SP1 Windows Server 2003 64-Bit Itanium SP1 Windows Server 2003 64-Bit Itanium Enterprise Edition SP1

Компонент	Требование
Программное обеспечение	Microsoft SQL Server Setup требует наличия Microsoft Windows Installer версии 3.1 или более поздней, а также компонентов Microsoft Data Access Components версии 2.8 SP1 или более поздней версии
Сетевые компоненты	Требования к сетевым компонентам одинаковы для 32-битных и 64-битных версий продукта. Необходимые компоненты входят в состав операционных систем Windows Server 2008, Windows Server 2003, Windows XP и Windows 2000. SQL Server не поддерживает следующие сетевые протоколы: ■ Banyan VINES Sequenced Packet Protocol (SPP) ■ Multiprotocol ■ AppleTalk ■ NWLink IPX/SPX Для подключения к SQL Server следует выбрать поддерживаемые продуктом протоколы. Отдельные экземпляры SQL Server, как именованные, так и с именами по умолчанию, поддерживают следующие сетевые протоколы: ■ Shared memory ■ Named pipes ■ TCP/IP ■ VIA Отметим, что Shared memory не поддерживается в отказоустойчивых кластерах
Интернет	Для установки SQL Server требуется Microsoft Internet Explorer 6 SP1 или более поздней версии. Этот компонент используется Microsoft Management Console (MMC), SQL Server Management Studio, Business Intelligence Development Studio, компонентом Report Designer из Reporting Services и HTML Help
Память	■ Минимум: 512 Мбайт ■ Рекомендуется 1 Гбайт или более ■ Максимум: зависит от операционной системы

64-битные версии SQL Server 2008

В следующей таблице показаны системные требования для 64-битной версии **SQL Server 2008 Developer Edition**.

Компонент	Требование
Процессор	Тип процессора: ■ Платформа IA64: минимум Itanium ■ Платформа x64: минимум AMD Opteron, AMD Athlon 64, Intel Xeon с поддержкой Intel EM64T, Intel Pentium IV с поддержкой EM64T Скорость процессора: ■ Платформа IA64: минимум 1 ГГц, рекомендуется 1 ГГц и выше ■ Платформа x64: минимум 1 ГГц, рекомендуется 1 ГГц и выше
.NET Framework	При установке SQL Server устанавливаются следующие компоненты, требуемые для обеспечения функционирования продукта: ■ Microsoft .NET Framework 2.0 ■ Microsoft SQL Server Native Client ■ Файлы, требующиеся Microsoft SQL Server Setup
Операционная система	Windows Vista 64-bit x64 Ultimate Edition Windows Vista 64-bit x64 Home Premium Edition Windows Vista 64-bit x64 Home Basic Edition Windows Vista 64-bit x64 Enterprise Edition Windows Vista 64-bit x64 Business Edition Windows Vista 64-bit x64 Ultimate Edition Windows Server 2008 64-bit x64 Standard Edition Windows Server 2008 64-bit x64 Data Center Edition Windows Server 2008 64-bit x64 Enterprise Edition Windows Server 2008 64-bit IA64 Itanium Edition Windows Server 2003 64-bit x64 SP1 Windows Server 2003 64-bit x64 Enterprise Edition SP1 Windows Server 2003 64-bit Itanium SP1 Windows Server 2003 64-bit Itanium Enterprise Edition SP1 Windows XP Professional 64-bit x64 SP2
Программное обеспечение	Microsoft SQL Server Setup требует наличия Microsoft Windows Installer версии 3.1 или более поздней, а также компонентов Microsoft Data Access Components версии 2.8 SP1 или более поздней версии
Сетевые компоненты	Требования к сетевым компонентам одинаковы для 32-битных и 64-битных версий продукта. Необходимые компоненты входят в состав операционных

Компонент	Требование
	систем Windows Server 2008, Windows Server 2003, Windows XP и Windows 2000. SQL Server не поддерживает следующие сетевые протоколы: ■ Banyan VINES Sequenced Packet Protocol (SPP) ■ Multiprotocol ■ AppleTalk ■ NWLink IPX/SPX Для подключения к SQL Server следует выбрать поддерживаемые продуктом протоколы. Отдельные экземпляры SQL Server, как именованные, так и с именами по умолчанию, поддерживают следующие сетевые протоколы: ■ Shared memory ■ Named pipes ■ TCP/IP ■ VIA Отметим, что Shared memory не поддерживается в отказоустойчивых кластерах
Интернет	Для установки SQL Server требуется Microsoft Internet Explorer 6 SP1 или более поздней версии. Этот компонент используется Microsoft Management Console (MMC), SQL Server Management Studio, Business Intelligence Development Studio, компонентом Report Designer из Reporting Services и HTML Help
Память	Платформа IA64: ■ Минимум: 512 Мбайт ■ Рекомендуется 1 Гбайт или более ■ Максимум: зависит от операционной системы Платформа X64: ■ Минимум: 512 Мбайт ■ Рекомендуется 1 Гбайт или более ■ Максимум: зависит от операционной системы

Требования к дисковому пространству

Во время установки SQL Server 2008 компонент Windows Installer создает ряд временных файлов на системном диске. Перед запуском программы Setup для установки или обновления SQL Server необходимо убедиться в том, что на системном жестком диске есть около 1.6 Гбайт свободного

пространства. Это требование распространяется даже на те сценарии, когда SQL Server устанавливается на диск, отличный от системного.

Дополнительные требования к дисковому пространству зависят от системной конфигурации и выбираемых для установки компонентов. В следующей таблице показаны требования к дисковому пространству основных компонентов SQL Server 2008.

Компонент	Требуемое дисковое пространство
Ядро базы данных и файлы данных, репликация и полнотекстовый поиск	280 Мбайт
Аналитика (Analysis Services) и файлы данных	90 Мбайт
Сервисы отчетов (Reporting Services) и Report Manager	120 Мбайт
Интеграционные сервисы (Integration Services)	120 Мбайт
Клиентские компоненты	850 Мбайт
Документация — SQL Server Books Online и SQL Server Compact 3.5 SP1 Books Online	240 Мбайт

Глава 1

Платформа для корпоративных данных

Новая версия Microsoft SQL Server — SQL Server 2008 представляет собой платформу для сбора, обработки и хранения корпоративных данных. SQL Server 2008 обеспечивает безопасность, надежность, а также оптимизированную и предсказуемую производительность при работе с данными любого объема.

К ключевым характеристикам продукта как безопасной и надежной платформы для корпоративных данных можно отнести следующие возможности: поддержка прозрачного для приложений шифрования данных — у разработчиков появляется возможность включения шифрования данных в уже существующие приложения без необходимости их переписывания, возможность внешнего управление ключами за счет консолидации ключей безопасности в центре обработки данных, новые функции поддержки ключевых механизмов аудита данных, подключаемые процессоры как один из способов добавления системных ресурсов без нарушения работы всей системы и выполняющихся на ней приложений и расширенные средства зеркалирования, включая прозрачную поддержку автоматического восстановления для уже существующих приложений.

С точки зрения предсказуемости и оптимизации производительности систем, построенных на основе SQL Server 2008, можно выделить следующие характеристики продукта: поддержка сбора и хранения данных о производительности, включая различные системные события, функции анализа производительности системы, поддержка сжатия данных, приводящая к снижению стоимости хранения информации и увеличению производительности запросов, новые модели оптимизации запросов и расширенные функции управления ресурсами системы, а также ее оптимальной загрузкой.

И, наконец, третья группа характеристик SQL Server 2008 относится к продуктивному управлению всей инфраструктурой, существенно снижающему затраты на выполнение рутинных операций. Сюда относится такая функциональность, как декларативное управление, основанное на политиках (policy), а не на сценариях (скриптовых программах), возможность управления всей серверной группой через задание политик на уровне всей компании с автоматическим мониторингом и применением этих политик, упрощенная установка и конфигурация, а также интеграция с системами корпоративного управления, включая возможность задания политик в соответствии с System Definition Model и управление данными и инфраструктурой через Microsoft System Center.

Рассмотрим некоторые характеристики SQL Server 2008 более подробно. Типовые вопросы, которые обычно встают перед администраторами баз данных, можно сформулировать следующим образом:

- Как обеспечить масштабируемость платформы без ущерба производительности?
- Как обеспечить полноценную доступность сервисов, связанных с предоставлением данных?
- Как наилучшим образом защитить данные?

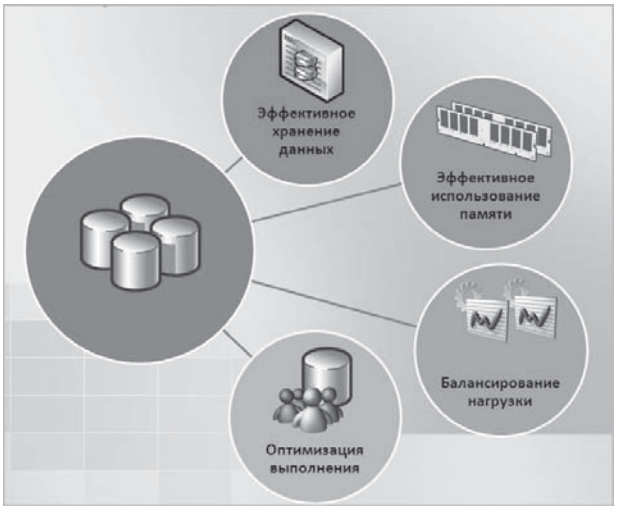
- Как наиболее оптимально управлять инфраструктурой баз данных?

Новая версия SQL Server помогает найти ответы на эти вопросы, предоставляя следующие ключевые возможности в области OLTP:

Ключевые возможности SQL Server 2008 в области OLTP

Масштабируемость для компаний любого размера

Масштабируемость для компаний любого размера — на базе SQL Server 2008 могут быть построены решения, отвечающие всем современным требованиям по обеспечению производительности и масштабируемости.



Масштабируемость для компаний любого размера

Требования к современным системам хранения данных включают поддержку масштабируемости и оптимального выполнения ключевых операций на

любых объемах данных. К фундаментальным возможностям SQL Server 2008 по обеспечению масштабируемости и производительности относятся:

- **Эффективное хранение данных** за счет полноценной, встроенной в ядро СУБД поддержки различных типов данных (включая типы данных xml, varchar(max), varbinary(max) и vardecimal), оптимизации способов хранения численных значений и нулевых данных, а также встроенных функций сжатия данных.
  - Ключевые характеристики:
    - эффективное хранение реляционных и нереляционных данных;
    - сжатие данных и резервных копий, снижающее требования к хранилищу и улучшающее производительность операций ввода вывода;
    - использование разреженных колонок для снижения места для хранения нулевых значений.
- **Эффективное использование** памяти за счет автоматического управления ресурсами и возможности масштабирования до объемов памяти, установленных в центрах обработки данных.
  - Ключевые характеристики:
    - автоматическое динамическое управление памятью;
    - снижение числа операций ввода/вывода за счет более эффективного управления буферами;
    - освобождение памяти по требованию;
    - поддержка больших объемов памяти — 64 Гбайт на Windows Server Datacenter Edition (используя AWE), до 8 Тбайт на 64-битной платформе.
- **Балансировка нагрузки** с использованием нового способа управления ресурсами — **Resource Governor**, с помощью которого появляется возможность проактивного управления использованием ресурсов.
  - Ключевые характеристики:
    - возможность задания границ и приоритетов для отдельных процессов и задач, выполняющихся на экземпляре SQL Server.

Средство управления ресурсами — **Resource Governor** — позволяет обеспечить более предсказуемую работу сервера под различными нагрузками (workloads). Это достигается за счет установки ограничений и присвоения приоритетов для определенных нагрузок. Нагрузки зависят от таких факторов, как пользователи, приложения и базы данных. Задание ограничений по ресурсам позволяет администраторам избежать таких ситуаций, как, например, монополизация ресурсов при выполнении неоптимизированных запросов.

Поддержка Resource Governor на уровне Data Definition Language и представлений реализована в SQL Server 2008 следующими объектами:

Управление пулом ресурсов	
CREATE RESOURCE POOL	Создает пул ресурсов
ALTER RESOURCE POOL	Изменяет конфигурацию пула ресурсов
ALTER RESOURCE POOL	Удаляет пул ресурсов
Управление загрузками	
CREATE WORKLOAD GROUP	Создание группы нагрузок и связь ее с пулом ресурсов
ALTER WORKLOAD GROUP	Изменение конфигурации группы нагрузок
DROP WORKLOAD GROUP	Удаление группы нагрузок
Управление Resource Governor	
ALTER RESOURCE GOVERNOR	Изменение конфигурации Resource Governor, включение и выключение Resource Governor, регистрация функций

Для конфигурирования пула ресурсов Resource Governor предоставляет четыре параметра: минимальный и максимальный процент использования процессора и минимальный и максимальный процент использования памяти. Для конфигурирования загрузки Resource Governor предоставляет следующие шесть параметров:

- Максимальный объем памяти для запроса;
- Максимальный процент использования процессора для запроса;
- Тайм-аут ресурсов для запроса;
- Относительная важность запроса;
- Максимальное число запросов для группы;
- Пул ресурсов, связанный с загрузкой.

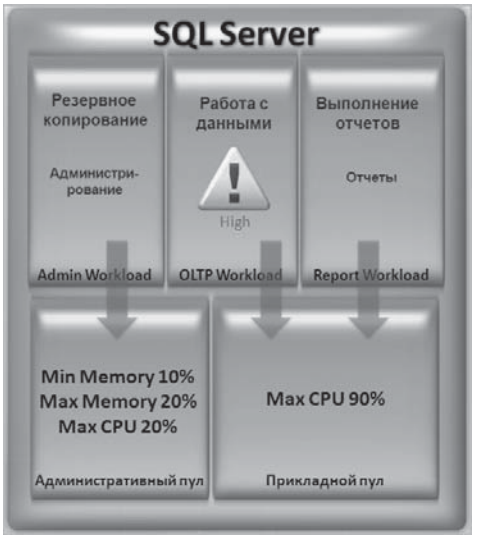
Для Resource Governor поддерживаются следующие представления:

sys.resource_governor_ configuration	Возвращает сохраненное состояние Resource Governor
sys.resource_governor_ resource_pools	Возвращает сохраненное состояние пула ресурсов. Каждая запись содержит конфигурацию отдельного пула
sys.resource_governor_ workload_groups	Возвращает сохраненное состояние загрузок
sys.dm_resource_governor_ workload_groups	Возвращает статистику для группы загрузок и текущую конфигурацию
sys.dm_resource_governor_ resource_pools	Возвращает данные о текущемсостоянии пула ресурсов, а также соответствующую статистику
sys.dm_resource_governor_ configuration	Возвращает текущую конфигурацию Resource Governor

Ряд представлений, поддерживавшихся в предыдущих версиях SQL Server, обновлен и возвращает информацию о Resource Governor:

sys.dm_exec_query_ memory_grants	Добавлены следующие колонки, относящиеся к Resource Governor: **group_id** **pool_id** **is_small** **ideal_memory_kb**
sys.dm_exec_query_resource_ semaphores	Добавлены следующие колонки, относящиеся к Resource Governor: **pool_id**
sys.dm_exec_sessions	Добавлены следующие колонки, относящиеся к Resource Governor: **group_id**
sys.dm_exec_requests	Добавлены следующие колонки, относящиеся к Resource Governor: **group_id**
sys.dm_exec_cached_plans	Добавлены следующие колонки, относящиеся к Resource Governor: **pool_id**
sys.dm_os_memory_brokers	Добавлены следующие колонки, относящиеся к Resource Governor: **pool_id** **allocations_kb_per_sec** **predicated_allocations_kb** **overall_limit_kb**
sys.dm_os_wait_stats	Возвращает информацию о задержках в выполняющихся потоках, которая может использоваться для обнаружения проблем, связанных с производительностью системы

- **Оптимизация выполнения параллельных заданий** через эффективное управление блокировками и изоляцию конфигурационных параметров.
  - Ключевые характеристики:
    - проактивное управление блокировками через опцию таблицы **LOCK_ESCALATION**;
    - уровни изоляции для поддержки всех сценариев параллельной работы;
    - более гранулярные блокировки на уровне записи, раздела, таблицы и базы данных.



Resource Governor

Существенное снижение времени простоя

Существенное снижение времени простоя — поддержка режима «always on», гарантируемая SQL Server, обеспечивает не только высокую доступность приложений, но и минимизирует затраты на управление.

Существенное снижение времени простоя

В SQL Server 2008 снижение времени простоя и, соответственно, повышение доступности сервера обеспечивается благодаря следующим технологиям, входящим в состав продукта:

- **Зеркалирование**, которое в данной версии более устойчиво и обеспечивает более высокую производительность. Зеркалирование обеспечивает автоматическую или ручную отказоустойчивость при сбое основного сервера. Для обеспечения дополнительного доступа к данным имеется возможность сконфигурировать зеркальный сервер таким образом, чтобы обеспечить доступ к зеркальной копии в режиме «только чтение». Возможными причинами сбоев могут быть проблемы с дисками, сбои в электропитании и т. п.

В SQL Server 2008 Enterprise Edition поддерживается целостность данных и возможность автоматической попытки восстановления при возникновении проблем со страницами данных, расположенными на диске. Такой подход позволяет SQL Server восстанавливаться после сбоев, связанных с физическим повреждением данных, более быстро и, при возможности, без участия администраторов. Реализация механизмов зеркалирования требует передачи log-информации (журнала транзакций) между серверами, участвующими в данном процессе. Передача больших объемов log-информации между серверами может вызвать нагрузку на сеть и привести к определенным задержкам, которые могут повлиять как на работу пользователей, так и других серверов.

Для оптимизации передачи данных в SQL Server 2008 используется сжатие потока исходящих log-данных, которое минимизирует нагрузку на сеть и снижает требования по скорости передачи и «толщине» канала. В большинстве сценариев поддержка механизмов зеркалирования не требует внесения каких-то изменений в клиентские приложения. В SQL Server 2008 обеспечивается прозрачное для клиента перенаправление на соответствующий сервер, что делает поддержку технологий зеркалирования возможной для широкого класса бизнес-приложений.

- **Отказоустойчивые кластеры** могут использоваться для обеспечения высокой доступности и защиты не только отдельных баз данных, но и всего экземпляра SQL Server. Для обеспечения такой возможности SQL Server 2008 может быть включен в группу, называемую Microsoft Cluster Service Cluster. Отказоустойчивый кластер выглядит для клиентов как единый экземпляр SQL Server, но обеспечивает переключение между серверами при возникновении сбоев на одном из серверов, включенных в кластер. Ряд ограничений, имевшихся в предыдущих версиях SQL Server, не позволял в полной мере использовать кластерные решения. К таким ограничениям относились требования использовать однобуквенное название диска для каждого экземпляра SQL Server и требование, чтобы все узлы в кластере находились в одной подсети. Кластеризация SQL Server при использовании Windows Server 2008 снимает эти ограничения и позволяет создавать более гибкие кластерные конфигурации. Помимо этого, в SQL Server 2008, работающем на платформе Windows Server 2008, поддерживаются кластера с числом до 16. Средство проверки кластеров (cluster validation tool) позволяет компаниям

выполнить проверку аппаратных конфигураций и выбрать наиболее подходящие для развертывания кластеров.

- **Репликация данных** — это возможность создания дополнительных копий базы данных, добавления копии данных для обращения в режиме «только чтение», например, для подсистемы отчетов или добавления базы данных в топологию репликации в режиме peer-to-peer, при котором изменения будут передаваться всем подключенным серверам.

Репликация в режиме peer-to-peer

- В SQL Server 2008 поддерживаются следующие типы репликаций: транзакционная (transactional), объединительная (merge) и snapshot-репликация для поддержки распределенных приложений. Мастер **Peer-to-Peer Topology Wizard** и утилита **Topology Viewer** позволяют администраторам баз данных более просто настраивать и конфигурировать транзакционные репликации в режиме peer-to-peer. Утилита **Topology Viewer** дает возможность получить визуальное изображение топологии; с помощью нового визуального дизайнера и мастера можно более просто создавать и модифицировать топологии, используемые при репликациях. Помимо этого, в предыдущих версиях SQL Server требовалась остановка сервера для добавления нового узла при репликации в режиме peer-to-peer. Сейчас этого не требуется — SQL Server 2008 позволяет добавлять новые узлы без остановки сервера: это позволяет критичным для обеспечения работоспособности бизнеса приложениям продолжать работать во время установки новых узлов, которые будут использоваться механизмами репликации.
- Онлайновые операции — это возможность SQL Server 2008, позволяющая выполнять управление индексами и операции восстановления данных без приостановки выполнения основных операций на сервере. Помимо этого, в редакции SQL Server 2008 Enterprise Edition поддерживается добавление на сервер, на котором работает SQL Server 2008, памяти и процессоров без необходимости в остановке самого сервера, что существенно повышает доступность SQL Server и обрабатываемых им данных и сокращает время нахождения сервера в режиме обслуживания (maintenance shutdown).

Т.н. «горячее» добавление памяти (**hot-add memory**) поддерживается при следующих условиях:

- Используется редакция SQL Server 2008 Enterprise Edition, 32-битная версия с включенной поддержкой AWE или 64-битная версия;
- SQL Server должен быть запущен с опцией командной строки **–h**;
- Используется платформа Windows Server 2003 Enterprise Edition или Windows Server 2003 Datacenter Edition;
- Аппаратная платформа должна поддерживать добавление памяти или необходимо использовать программное обеспечение для виртуализации.

Т. н. «горячее» добавление процессоров (**hot-add CPUs**) поддерживается при следующих условиях:

- Используется редакция SQL Server 2008 Enterprise Edition, 64-битная версия;
- Используется платформа Windows Server 2008 Enterprise Edition для систем на базе Itanium или Windows Server 2008 Datacenter Edition для систем x64;
- Аппаратная платформа должна поддерживать добавление процессоров или необходимо использовать программное обеспечение для виртуализации.

Отметим, что SQL Server не начинает автоматически использовать новые процессоры после их горячей установки — чтобы процессоры стали доступны SQL Server, необходимо выполнить команду **RECONFIGURE**.

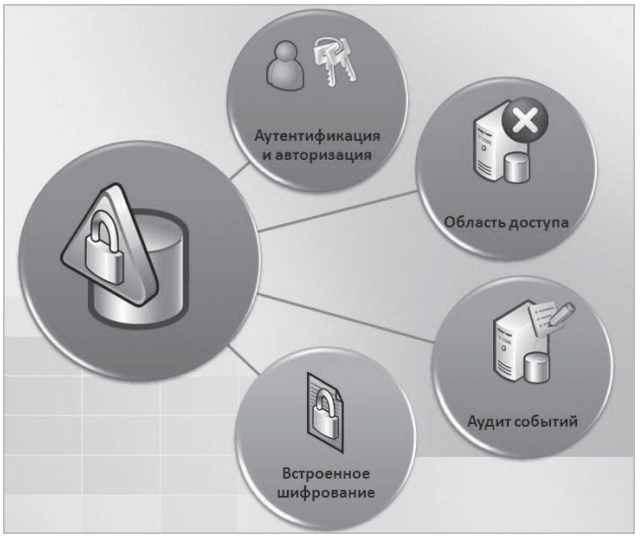
Защита критичных для бизнеса данных

Защита критичных для бизнеса данных — SQL Server 2008 представляет собой платформу с расширенными функциями безопасности, обеспечивая такие возможности, как шифрование данных, аудит изменений данных и метаданных, управление внешними крипто-ключами, шифрование и подписание файлов резервных копий.

Защита критичных для бизнеса данных обеспечивается в SQL Server 2008 на уровне следующих технологий и компонентов.

- **Расширенная модель аутентификации и авторизации**. В SQL Server 2005 поддерживалась возможность аутентификации пользователей базы данных через учетные записи Windows, учетные записи на уровне SQL Server, а также были реализованы механизмы для применения парольных настроек для всех пользователей. Администраторы баз данных могут использовать иерархическую модель присвоения привилегий для защиты данных и ресурсов на уровне сервера, базы данных или схемы.
- **Возможность конфигурации областей доступа и конфигурации «защита по умолчанию» («secure by default»)**. В начальном состоянии, сразу же после установки в SQL Server включены только те сер-

висы и функции, которые необходимы для выполнения базовых операций — дополнительные сервисы и функции отключены. К отключенным сервисам и функциям, например, относятся: интегрированная поддержка CLR, зеркалирование, отладка, Service Broker, SQL Mail и ряд других. Эти сервисы установлены в системе, но не запущены и не доступны до тех пор, пока администраторы не включат их или не сконфигурируют соответствующим образом. Администраторы могут централизованно управлять всеми сервисами и функциями, используя специальную утилиту конфигурации (**Surface Configuration Tool**) — это позволяет снизить область атаки и активизировать только те сервисы и функции, которые реально необходимы для обеспечения работы того или иного приложения.



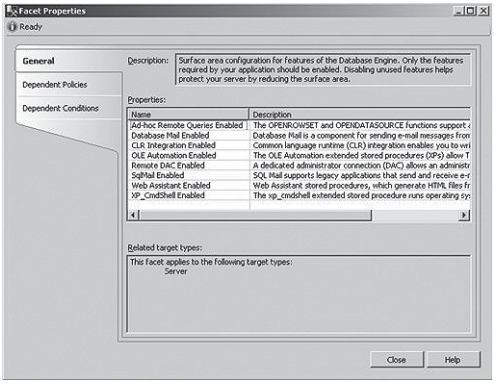
Защита критичных для бизнеса данных

Без использования упомянутой выше утилиты, практически единственный способ включения дополнительных сервисов и функций — это использование системной хранимой процедуры **sp_configure**, которая может быть вызвана следующим образом:

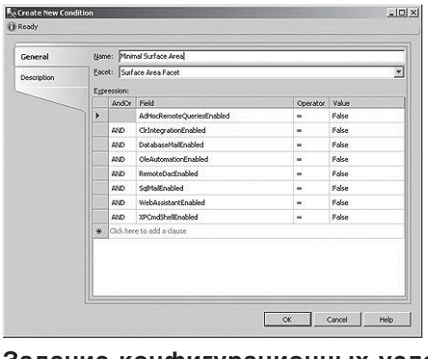
```
sp_configure 'show advanced options', 1
reconfigure with override
sp_configure 'clr enabled', 1
```

Обнаружение правильного синтаксиса для включения или выключения ряда сервисов и функций может стать достаточно продолжительной задачей, поэтому для решения этих задач в SQL Server 2008 рекомендуется использовать либо упомянутую выше утилиту конфигурации, либо политики на уровне Declarative Management Framework (DMF). DMF поддержи-

вает ряд «экранов» (Facets), каждый из которых содержит набор конфигурационных опций. Такие экраны можно использовать для описания условий, содержащих требуемые настройки и применить эти настройки, используя поддержку политик в SQL Server 2008. Одним из таких экранов является экран **Surface Area**, с помощью которого задается состояние различных сервисов и функций SQL Server 2008.



Экран Surface Area



Задание конфигурационных условий

- **Полный аудит всех событий, происходящих в SQL Server**. В SQL Server 2008 поддерживается возможность многоуровневого аудита – как на уровне всего сервера, так и на уровне базы данных. Использование аудита позволяет ответить на такие вопросы, как «Кто и когда модифицировал эти данные?» или «Сколько было неудачных попыток подключения к системе?». В дополнение к гибким возможностям по конфигурированию различных уровней аудита, поддерживается возможность переноса настроек аудита с сервера на сервер, что существенно упрощает развертывание и настройку системы аудита на корпоративном уровне. При развертывании механизмов аудита на уровне компании, механизм **Auditing Data Collector** консолидирует отчеты и предоставляет средства для анализа собранных данных.

Поддержка аудита в SQL Server 2008 реализована на уровне объекта **Audit**, который позволяет администраторам собирать активности, происходившие в базе данных и сохранять их в протоколе. В SQL Server 2008 поддерживается возможность сохранения данных аудита в:

- файлах;
- протоколе Windows Application Log;
- протоколе Windows Security Log.

Для записи данных в протокол Windows Security Log сервис SQL Server должен быть сконфигурирован как Local System, Local Service, Network Service или как доменная учетная запись с привилегией **SeAuditPrivilege**. Для создания объекта **Audit** используется команда **CREATE SERVER AUDIT.** Она задает объект **Audit** и ассоциирует его с указанным объектом на уровне сервера. Дополнительные опции данной команды используются для конфигурирования объекта **Audit** в зависимости от указанного серверного объекта. В следующем примере показано, как создать два объекта **Audit** — один для записи протокола в файле, второй — для записи активностей в протокол Windows Application Log.

```
CREATE SERVER AUDIT File_Audit
    TO FILE ( FILEPATH='\\SQLPROD_1\Audit\' );

CREATE SERVER AUDIT AppLog_Audit
    TO APPLICATION_LOG
    WITH ( QUEUE_DELAY = 500,  ON_FAILURE = SHUTDOWN);
```

После того как объект **Audit** создан, можно начать добавлять к нему события, используя команды **CREATE SERVER AUDIT SPECIFICATION** и **CREATE DATABASE AUDIT SPECIFICATION**. Команда **CREATE SERVER AUDIT SPECIFICATION** используется для добавления событий, которые происходят на уровне сервера. Ниже показан пример добавления группы событий **FAILED_LOGIN_GROUP** к созданному ранее объекту **File_Audit**:

```
CREATE SERVER AUDIT SPECIFICATION Failed_Login_Spec
FOR SERVER AUDIT File_Audit
    ADD (FAILED_LOGIN_GROUP);
```

Команда **CREATE DATABASE AUDIT SPECIFICATION** используется для добавления событий, которые происходят на уровне базы данных. Ниже показан пример добавления группы событий **DATABASE_OBJECT_CHANGE_GROUP**, к которым относятся такие операции, как **CREATE**, **ALTER** и **DROP** на уровне базы данных и команды **INSERT**, **UPDATE** и **DELETE**, выполняемые над объектами в схеме **Sales** пользователям с учетной записью **SalesUser** или **SalesAdmin** к созданному ранее объекту A**ppLog_Audit**:

```
CREATE DATABASE AUDIT SPECIFICATION Sales_Audit_Spec
FOR SERVER AUDIT AppLog_Audit
```

```
ADD (DATABASE_OBJECT_CHANGE_GROUP),
ADD (INSERT, UPDATE, DELETE
     ON Schema::Sales
     BY SalesUser, SalesAdmin);
```

- **Встроенная поддержка прозрачного шифрования данных**. В SQL Server 2008 поддерживается механизм прозрачного шифрования данных (transparent data encryption, TDE), который может быть включен на уровне базы данных для шифрования всей базы или файла с данными без необходимости в изменении клиентских приложений. Данные шифруются и расшифровываются при записи и считывании с диска. Такой подход позволяет создавать индексы и выполнять поиск (включая полнотекстовый поиск) по зашифрованным данным без использования каких-либо дополнительных функций.

Прозрачное шифрование данных

Для использования прозрачного шифрования данных необходимо выполнить следующие шаги:

- Создать мастер-ключ;
- Создать или получить сертификат, защищенный мастер-ключом;
- Создать ключ для шифрования базы данных и защитить его сертификатом;
- Задать шифрование на уровне базы данных.

В следующем примере показано, как выполнить шифрование и расшифровку базы данных **AdventureWorks**, используя сертификат, установленный на сервере:

```
 USE master;
GO
CREATE MASTER KEY ENCRYPTION BY PASSWORD = '<UseStrongPasswordHere>';
GO
```

```
CREATE CERTIFICATE MyServerCert WITH SUBJECT = 'My DEK Certificate'
GO

USE AdventureWorks
GO

CREATE DATABASE ENCRYPTION KEY
WITH ALGORITHM = AES_128
ENCRYPTION BY SERVER CERTIFICATE MyServerCert
GO

ALTER DATABASE AdventureWorks
SET ENCRYPTION ON
GO
```

Операции шифрования и расшифровки выполняются как фоновые потоки SQL Server. Для просмотра состояния этих операций можно использовать представления, список которых приведен ниже. Для восстановления зашифрованной базы данных требуется доступ к сертификату или ассиметричному ключу, который использовался для шифрования базы. Без сертификата или ключа база данных не может быть восстановлена. В результате, сертификат, который использовался для шифрования ключа базы данных, должен сохраняться вместе с резервной копией базы.

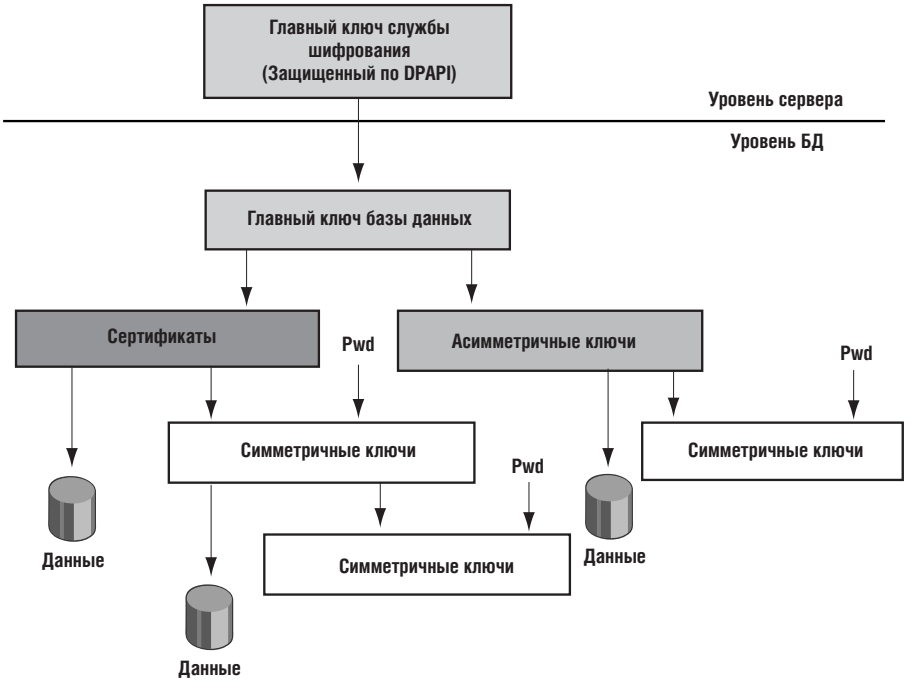
В SQL Server 2008 поддерживаются следующие команды и функции для шифрования баз данных:

Команда/Функция	Описание
CREATE DATABASE ENCRYPTION KEY	Создает ключ, использующийся для шифрования базы
ALTER DATABASE ENCRYPTION KEY	Изменяет ключ, использующийся для шифрования базы
DROP DATABASE ENCRYPTION KEY	Удаляет ключ, использующийся для шифрования базы

Следующие представления используются для прозрачного шифрования базы данных:

Представление	Описание
sys.databases	Получение информации о базах данных
sys.certificates	Получение списка сертификатов в базе данных
sys.dm_database_encryption_keys	Получение информации о ключах шифрования, используемых в базе данных и о состоянии шифрования на уровне базы

- Помимо встроенных средств управления ключами, впервые появившихся в SQL Server 2005, в SQL Server 2008 поддерживается использование крипто-провайдеров и систем управления ключами сторонних производителей, а также использование аппаратных систем шифрования (Hardware Security Modules, HSM).



Иерархия ключей шифрования в SQL Server 2008

- Такой подход позволяет объединить механизмы шифрования и управления ключами, используемые в различных приложениях и сервисах на уровне организации, в единое целое и тем самым упростить их настройку и управление.

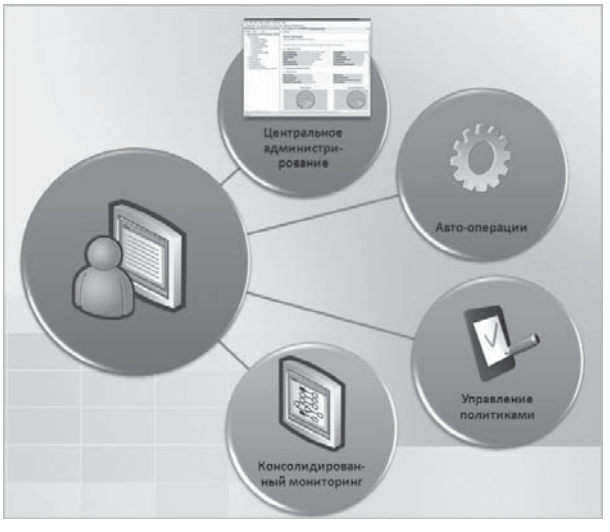
Снижение затрат на администрирование

Снижение затрат на администрирование — использование администрирования на основе политик с автоматической проверкой их внедрения, наличие улучшенных средств анализа производительности, обнаружения и исправления ошибок и сбоев, а также расширенные механизмы настройки системы.

В SQL Server 2008 поддерживаются различные средства автоматизации администрирования, к которым, в частности, относятся:

- **Централизованное администрирование с использованием единого интерфейса**. Входящая в состав SQL Server 2008 утилита SQL Server

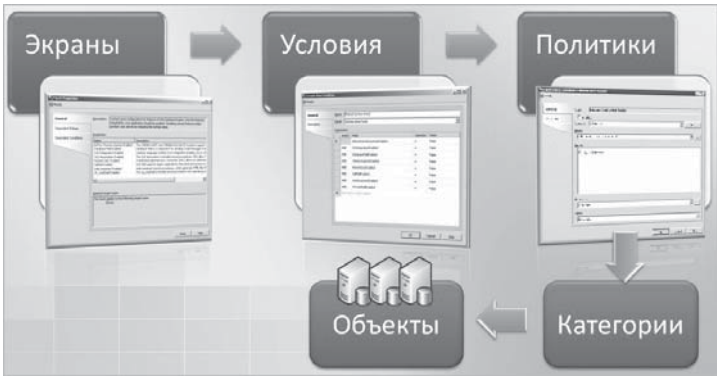
Management Studio предоставляет в распоряжение администраторов интегрированную среду для доступа, управления и конфигурирования баз данных и как локальных, так и удаленных серверов.



Снижение затрат на администрирование

- **Автоматизация операций, связанных с сопровождением и обслуживанием баз данных и быстрой реакции на возникающие проблемы**. В состав SQL Server 2008 входит SQL Server Agent — компонент, позволяющий автоматизировать комплексные задачи, связанные с обслуживанием и сопровождением серверов баз данных во всей организации. Система управления оповещениями и возможности с заданием расписаний для выполнения определенных задач позволяют освободить администраторов от необходимости выполнения множества рутинных операций, требуемых для поддержания работоспособности корпоративных решений.
- **Управление на уровне политик с использованием Declarative Management Framework (DMF)**. Поддержка Declarative Management Framework в SQL Server 2008 позволяет задавать конфигурации на уровне политик (policy) и применять их к серверам, базам данных, таблицам и другим объектам корпоративных баз данных. Такие политики могут как запрещать изменения в системе, так и выполнять мониторинг определенных изменений. В SQL Server 2008 предоставляется возможность использования триггеров для принудительного выполнения политик, Service Broker для обработки после применения политик или SQL Server Agent для задания расписания применения политик. В SQL Server 2008 также поддерживается выполнение в реальном времени проверки соответствия определенных объектов заданным политикам.

Как мы уже упомянули выше, использование DMF состоит в использовании специальных экранов (facets) для задания условий, которые затем описываются политиками, принадлежащими к определенным категориями. После того как политики определены, они применяются к соответствующим объектам.



DMF предполагает использование следующих типов объектов при конфигурации политик:

- **Экраны (Facets)** — содержат свойства, соответствующие опциям, относящимся к группе, управляемой политикой. Например, Surface Area включает свойства, относящиеся к определенным возможностям SQL Server — таким, как **Database Mail Enabled** или **CLR Integration Enabled**;
- **Условия (Conditions)** — состояние экрана. Основываются на одном экране и могут использоваться в одной или более политик. Например, можно создать условие с названием **Minimal Surface Area**, в котором будут собраны все свойства экрана **Surface Area**, имеющие значение **False**;
- **Политики (Policies)** — содержат условие, которое должно быть выполнено для одного или более объектов. Например, можно создать политику с названием **Locked Down Server**, которая присваивает серверу условие **Minimal Surface Area**;
- **Категории (Categories)** — содержат одну или более политик. Например, можно создать категорию с именем **Corporate DB Policies**, в которой будут указаны политики именования объектов базы данных, политики задания уровня совместимости базы, а затем подписать базу на эту категорию. Помимо этого, категории могут быть активными или пассивными на уровне сервера или базы данных, что позволяет администраторам управлять применением политик;
- **Объекты (Targets)** — сервера, базы данных, учетные записи, таблицы и другие объекты базы данных, с которыми связаны политики.

Для получения информации о политиках, условиях, группах, фильтрах и т. п., в SQL Server 2008 используются следующие представления:

syspolicy_conditions	syspolicy_policy_group_subscriptions
syspolicy_policies	syspolicy_policy_groups
syspolicy_policy_execution_history	syspolicy_system_health_state
syspolicy_policy_execution_history_details	syspolicy_target_filters

- **Консолидированный мониторинг с использованием нового средства — Performance Studio**. Набор средств для обнаружения и исправления проблем, связанных с производительностью системы, в состав SQL Server 2008 входит набор средств для настройки, мониторинга состояния и исправления проблем — **Performance Studio**. Данное средство включает три ключевых функции — сбор низкоуровневых данных от различных источников — SQL Trace, System Monitor, Dynamic Management Views и протокола, централизованное хранилище собранных данных с возможностью хранения исторических данных, базовых конфигураций и выполнения сравнений между различными конфигурациями, а также интегрированные средства генерации отчетов на базе Reporting Services для отображения различной информации и помощи администраторам в нахождении причин возникших проблем и анализа возможных вариантов их устранения.

Заключение

В данном разделе мы познакомились с тем, как Microsoft SQL Server 2008 обеспечивает основные характеристики, относящиеся к платформе для сбора, обработки и хранения корпоративных данных — безопасность, надежность, а также оптимизированную и предсказуемую производительность при работе с данными любого объема.

Далее мы рассмотрим ряд новинок, появившихся в SQL Server 2008 на уровне языка T-SQL.

Глава 2

Новая функциональность в T-SQL

В данном разделе мы рассмотрим ряд новинок, появившихся в SQL Server 2008 на уровне языка Transact-SQL. К таким новинкам относятся:

- Типы данных Date и Time;
- Конструкция MERGE;
- Табличные параметры;
- Определение зависимостей объектов;
- Расширения на уровне SQL/CLR;
- Расширения DDL-триггеров;
- Расширения для GROUP BY;
- Различные улучшения.

Типы данных Date и Time

В SQL Server 2008 поддерживаются следующие типы данных для представления даты и времени: **Date**, **Time**, **DateTimeOffset**, **DateTime2**.

Тип данных **Date** используется для представления даты и поддерживает даты по Грегорианскому календарю в диапазонах от 0001-01-01 до 9999-01-01 при использовании формата ГГГГ-ММ-ДД. Дата хранится с точностью до одного дня, занимает 3 байта и не содержит данных о часовом поясе. Пример использования данного типа показан ниже.

```
DECLARE @MyDate date
CREATE TABLE Table1 (Column1 date)
```

Тип данных **Time** служит для хранения времени в диапазоне от 00:00:00.0000000 до 23:59:59.9999999 с точностью до 100 наносекунд, занимает от 3 до 5 байт и не содержит данных о часовом поясе. Пример использования данного типа показан ниже.

```
DECLARE @MyTime time(7)
CREATE TABLE Table1 (Column1 time(7))
```

Параметр при описании типа — 7 в данном примере — указывает на точность представления дробной части числа секунд.

Тип данных **DateTimeOffset** используется для задания даты и времени с информацией о часовом поясе и поддержкой 24-часового представления времени. Дата представляется в диапазоне от 0001-01-01 до 9999-01-01, время — от 00:00:00.0000000 до 23:59:59.9999999, часовой пояс — от -14:00 до +14:00. Пример использования данного типа показан ниже.

```
DECLARE @MyDatetimeoffset datetimeoffset(7)
CREATE TABLE Table1 (Column1 datetimeoffset(7))
```

Параметр при описании типа — 7 в данном примере — указывает на точность представления дробной части числа секунд.

Тип данных **DateTime2** представляет собой расширенный вариант типа **DateTime** и используется для представления более широкого диапазона дат без учета часового пояса.

```
DECLARE @MyDatetime2 datetime2(7)
CREATE TABLE Table1 (Column1 datetime2(7))
```

Параметр при описании типа — 7 в данном примере — указывает на точность представления дробной части числа секунд.

Ниже показан пример использования рассмотренных нами типов для представления даты и времени

```
CREATE TABLE t1 (c1 DATE, c2 TIME(3),
    c3 DATETIME2(7) NOT NULL DEFAULT GETDATE(),
    c4 DATETIMEOFFSET CHECK (c4 < CAST(GETDATE() AS
    DATETIMEOFFSET(0)))
);
INSERT INTO t1 VALUES ('0001-01-01', '23:59:59',
    '0001-12-21 23:59:59.1234567',
    '0001-10-21 23:59:59.1234567 -07:00');
INSERT INTO t1 VALUES ('9999-12-31', '23:59:59',
    '9999-12-31 23:59:59.1234567',
    '1111-10-21 23:59:59.1234567 -07:00');
SELECT c4, DATEPART(TZOFFSET, c4),
    DATEPART(ISO_WEEK, c4),
    DATEPART(MICROSECOND, c4)
FROM t1;
```

В языке запросов T-SQL поддерживается набор функций для работы с датой и временем. Некоторые их этих функций перечислены в следующей таблице.

Функция	Описание
SYSDATETIME	Возвращает текущие дату и время без учета часового пояса
SYSDATETIMEOFFSET	Возвращает текущие дату и время с учетом часового пояса
SYSUTCDATETIME	Возвращает текущие дату и время с учетом в формате UTC
CURRENT_TIMESTAMP	Возвращает текущее время
GETDATE	Возвращает текущее время
GETUTCDATE	Возвращает текущее время в формате UTC
DATENAME	Возвращает символьное обозначение даты

Функция	Описание
DATEPART	Возвращает указанную часть даты
DAY	Возвращает день для заданной даты
MONTH	Возвращает месяц для заданной даты
YEAR	Возвращает год для заданной даты
DATEDIFF	Возвращает разницу между двумя датами
DATEADD	Возвращает новое значение, полученное прибавлением указанного интервала к дате
SWITCHOFFSET	Изменяет часовой пояс
TODATETIMEOFFSET	Преобразует дату и время в UTC-формат

Конструкция MERGE

Новая конструкция **MERGE** позволяет эффективно выполнять несколько DML-команд (INSERT/UPDATE/DELETE), которые выполняются над одной таблицей в рамках одного запроса.

Возможные сценарии использования данной конструкции — это вставка или обновление данных для OLTP-операций, вставка или обновление данных для операций в рамках хранилищ данных, синхронизация двух баз данных и ряд других.

Рассмотрим использование конструкции **MERGE** для выполнения операций INSERT и UPDATE. Предположим, что в хранилище данных есть таблица **FactBuyingHabits**, в которой сохранена дата последней покупки для каждого клиента. Во второй таблице — **PurchaseRecords** — хранятся записи о покупках за неделю. Каждую неделю мы хотим добавлять записи о продуктах, которые никто ранее не покупал из таблицы **PurchaseRecords** в таблицу **FactBuyingHabits**. Для клиентов, которые повторно покупали товары, мы просто хотим обновлять дату покупки в таблице **FactBuyingHabits**. Операции **INSERT** и **UPDATE** могут быть выполнены с помощью одной операции **MERGE**.

Сначала мы создаем таблицы **FactBuyingHabits** и **PurchaseRecords** и загружаем в них данные. Так как операции **MERGE** наиболее оптимально работают при наличии индексов для объединенных ключей, мы создаем кластерные индексы для обеих таблиц.

```
IF OBJECT_ID (N'dbo.PurchaseRecords', N'U') IS NOT NULL
DROP TABLE dbo.PurchaseRecords;
GO

CREATE TABLE PurchaseRecords (ProductID int, CustomerID int, PurchaseDate datetime);
GO
```

```
IF EXISTS (SELECT name FROM sys.indexes
    WHERE name = N'IX_PurchaseRecords_ProductID')
    DROP INDEX IX_PurchaseRecords_ProductID ON dbo.PurchaseRecords;

GO

CREATE CLUSTERED INDEX IX_PurchaseRecords_ProductID
    ON dbo.PurchaseRecords (ProductID);
GO

INSERT INTO PurchaseRecords VALUES(707, 11794, '20060821'),
(707, 15160, '20060825'),(708, 18529, '20060821'),
(711, 11794, '20060821'),(711, 19585, '20060822'),
(712, 14680, '20060825'),(712, 21524, '20060825'),
(712, 19072, '20060821'),(870, 15160, '20060823'),
(870, 11927, '20060824'),(870, 18749, '20060825');
GO

IF OBJECT_ID (N'dbo.FactBuyingHabits', N'U') IS NOT NULL
DROP TABLE dbo.FactBuyingHabits;
GO

CREATE TABLE FactBuyingHabits (ProductID int, CustomerID int, LastPurchaseDate
datetime);
GO

IF EXISTS (SELECT name FROM sys.indexes
    WHERE name = N'IX_FactBuyingHabits_ProductID')
    DROP INDEX IX_FactBuyingHabits_ProductID ON dbo.FactBuyingHabits;
GO

CREATE CLUSTERED INDEX IX_FactBuyingHabits_ProductID
    ON dbo.FactBuyingHabits (ProductID);
GO

INSERT INTO FactBuyingHabits VALUES(707, 11794, '20060814'),
(707, 18178, '20060818'),(864, 14114, '20060818'),
(866, 13350, '20060818'),(866, 20201, '20060815'),
(867, 20201, '20060814'),(869, 19893, '20060815'),
(870, 17151, '20060818'),(870, 15160, '20060817'),
(871, 21717, '20060817'),(871, 21163, '20060815'),
(871, 13350, '20060815'),(873, 23381, '20060815');
GO
```

После выполнения приведенных выше операций, таблицы будут содержать следующие данные:

ProductID	CustomerID	PurchaseDate
707	11794	2006-08-20 00:00:00.000
707	15160	2006-08-25 00:00:00.000
708	18529	2006-08-21 00:00:00.000
711	11794	2006-08-20 00:00:00.000
711	19585	2006-08-22 00:00:00.000
712	14680	2006-08-26 00:00:00.000
712	21524	2006-08-26 00:00:00.000
712	19072	2006-08-20 00:00:00.000
870	15160	2006-08-23 00:00:00.000
870	11927	2006-08-24 00:00:00.000
870	18749	2006-08-25 00:00:00.000

Данные в таблице PurchaseRecords

ProductID	CustomerID	LastPurchaseDate
707	11794	2006-08-14 00:00:00.000
707	18178	2006-08-18 00:00:00.000
864	14114	2006-08-18 00:00:00.000
866	13350	2006-08-18 00:00:00.000
866	20201	2006-08-15 00:00:00.000
867	20201	2006-08-14 00:00:00.000
869	19893	2006-08-15 00:00:00.000
870	17151	2006-08-18 00:00:00.000
870	15160	2006-08-17 00:00:00.000
871	21717	2006-08-17 00:00:00.000
871	21163	2006-08-15 00:00:00.000
871	13350	2006-08-15 00:00:00.000
873	23381	2006-08-15 00:00:00.000

Данные в таблице FactBuyingHabits

Обратим внимание на тот факт, что две записи Product-Customer одинаковы для двух таблиц: клиент с идентификатором 11794 купил продукт 707 в течение недели, а также на предыдущей неделе; то же самое верно для клиента 15160 и продукта 870. Для этих записей мы обновим таблицу **FactBuyingHabits** датой покупок из **PurchaseRecords**, используя конструкцию **WHEN NOT MATCHED THEN**:

```
MERGE FactBuyingHabits AS fbh
USING (SELECT CustomerID, ProductID, PurchaseDate
FROM PurchaseRecords) AS src
ON (fbh.ProductID = src.ProductID AND
 fbh.CustomerID = src.CustomerID)
WHEN MATCHED THEN
    UPDATE SET fbh.LastPurchaseDate = src.PurchaseDate
WHEN NOT MATCHED THEN
    INSERT (CustomerID, ProductID, LastPurchaseDate)
    VALUES (src.CustomerID, src.ProductID, src.PurchaseDate);
```

Выполнение операций **UPDATE** и **DELETE** также можно осуществить с помощью конструкции **MERGE**. В следующем примере показано использование конструкции **MERGE** для ежедневного обновления таблицы **ProductInventory** в базе данных **AdventureWorks** на основании обработки таблицы **SalesOrderDetail**. Используя показанную ниже конструкцию **MERGE**, колонка **Quantity** таблицы **ProductInventory** обновляется вычитанием числа заказов, полученных каждый день для каждого продукта.

Если в результате этой операции число продуктов дойдет до нуля, запись о продукте удаляется из таблицы ProductInventory. Обратите внимание на то, что исходная таблица агрегируется по колонке **ProductID**. Если этого не сделать, может произойти ошибка при выполнении конструкции **MERGE** из-за того, что будет найден более чем один идентификатор продукта, соответствующий критерию.

```
USE AdventureWorks;
GO
MERGE Production.ProductInventory AS pi
USING (SELECT ProductID, SUM(OrderQty) FROM Sales.SalesOrderDetail sod
    JOIN Sales.SalesOrderHeader soh
    ON sod.SalesOrderID = soh.SalesOrderID
    AND soh.OrderDate = GETDATE()
    GROUP BY ProductID) AS src (ProductID, OrderQty)
ON (pi.ProductID = src.ProductID)
WHEN MATCHED AND pi.Quantity - src.OrderQty = 0
    THEN DELETE;
WHEN MATCHED
    THEN UPDATE SET pi.Quantity = pi.Quantity - src.OrderQty
```

Табличные параметры

Табличные параметры (Table-Valued Parameters) — это пользовательский тип для табличных данных, позволяющий использовать более одного параметра в запросах, передавать табличные данные функциям и эффективно обмениваться данными между клиентом.

Табличные данные могут использоваться, например, для передачи больших объемов данных с клиента на сервер или для применения бизнес-логики перед обновлением данных.

Ниже приведен пример использования табличных параметров — создание параметра соответствующего типа, объявление переменной для ссылки на этот параметр, заполнение списка параметров и передача значений хранимой процедуре.

```
USE AdventureWorks;
GO

/*
Создание табличного типа
*/
CREATE TYPE LocationTableType AS TABLE
(LocationName VARCHAR(50), CostRate INT);
GO
```

```
/*
Создание процедуры для получения данных
*/
CREATE PROCEDURE usp_InsertProductionLocation
    @TVP LocationTableType READONLY
    AS
    SET NOCOUNT ON
    INSERT INTO [AdventureWorks].[Production].[Location]
           ([Name]
           ,[CostRate]
           ,[Availability]
           ,[ModifiedDate])
        SELECT *, 0, GETDATE()
        FROM  @TVP;
        GO

/*
Объявление переменной для ссылки на тип
*/
DECLARE @LocationTVP
AS LocationTableType;

/*
Добавление данных к переменной
*/
INSERT INTO @LocationTVP (LocationName, CostRate)
    SELECT [Name], 0.00
    FROM
    [AdventureWorks].[Person].[StateProvince];

/*
Передача табличной переменной хранимой процедуре
*/
EXEC usp_InsertProductionLocation @LocationTVP;
GO
```

Определение зависимостей объектов

Механизм определения зависимостей между объектами позволяет получать необходимые данные для хранимых процедур, таблиц, представлений, функций, триггеров, пользовательских типов и т. п. Этот механизм работает как для объектов, привязанных к схеме, так и для других типов объектов.

Можно привести следующие сценарии использования механизма определения зависимостей между объектами:

- Нахождение всех объектов, от которого зависит данный объект;
- Нахождение всех объектов, зависящих от данного;
- Нахождение всех объектов, зависящих от другой базы данных;
- Нахождение всех объектов, выполняющих распределенные запросы.

Несколько новых системных представлений позволяют реализовать механизм определения зависимостей между объектами в SQL Server 2008. Среди этих представлений:

- sys.sql_expression_dependencies
  - Представление, пришедшее на смену **sys.sql_dependencies**;
  - Отслеживает зависимости как для объектов, привязанных к схеме, так и для других типов объектов;
  - Отслеживает зависимости между базами данных и между серверами.

Пример использования данного представления показан ниже — мы возвращаем все таблицы и колонки, используемые в представлении **Production.vProductAndDescription**. Данное представление зависит от объектов (таблиц и колонок), возвращаемых в колонках **referenced_entity_name и referenced_column_name**.

```
USE AdventureWorks;
GO
SELECT OBJECT_NAME(referencing_id) AS referencing_entity_name,
    o.type_desc AS referencing_desciption,
    COALESCE(COL_NAME(referencing_id, referencing_minor_id), '(n/a)') AS
referencing_minor_id,
    referencing_class_desc, referenced_class_desc,
    referenced_server_name, referenced_database_name, referenced_schema_name,
    referenced_entity_name,
    COALESCE(COL_NAME(referenced_id, referenced_minor_id), '(n/a)') AS
referenced_column_name,
    is_caller_dependent, is_ambiguous
FROM sys.sql_expression_dependencies AS sed
INNER JOIN sys.objects AS o ON sed.referencing_id = o.object_id
WHERE referencing_id = OBJECT_ID(N'Production.vProductAndDescription');
GO
```

Представление **sys.sql_expression_dependencies** также можно использовать для нахождения объектов, ссылающихся на другие объекты. Ниже показан пример кода, возвращающего объекты, ссылающиеся на таблицу **Production.Product** — эти объекты будут занесены в колонку **referencing_entity_name**.

```
USE AdventureWorks;
GO
SELECT OBJECT_SCHEMA_NAME ( referencing_id ) AS referencing_schema_name,
    OBJECT_NAME(referencing_id) AS referencing_entity_name,
    o.type_desc AS referencing_desciption,
    COALESCE(COL_NAME(referencing_id, referencing_minor_id), '(n/a)') AS
referencing_minor_id,
    referencing_class_desc, referenced_class_desc,
    referenced_server_name, referenced_database_name, referenced_schema_name,
    referenced_entity_name,
    COALESCE(COL_NAME(referenced_id, referenced_minor_id), '(n/a)') AS
referenced_column_name,
    is_caller_dependent, is_ambiguous
FROM sys.sql_expression_dependencies AS sed
INNER JOIN sys.objects AS o ON sed.referencing_id = o.object_id
WHERE referenced_id = OBJECT_ID(N'Production.Product');
GO
```

И, наконец, представление **sys.sql_expression_dependencies** можно использовать для нахождения перекрестных зависимостей между базами данных. В следующем примере показано, как получить все перекрестные зависимости для базы данных. Сначала мы создаем базу данных db1 и две хранимые процедуры, которые ссылаются на таблицы в базах данных db2 и db3. Затем мы обращаемся к представлению для нахождения перекрестных зависимостей между процедурами и таблицами.

```
CREATE DATABASE db1;
GO
USE db1;
GO
CREATE PROCEDURE p1 AS SELECT * FROM db2.s1.t1;
GO
CREATE PROCEDURE p2 AS
    UPDATE db3..t3
    SET c1 = c1 + 1;
GO
SELECT OBJECT_NAME (referencing_id),referenced_database_name,
    referenced_schema_name, referenced_entity_name
FROM sys.sql_expression_dependencies
WHERE referenced_database_name IS NOT NULL;
GO
USE master;
GO
DROP DATABASE db1;
GO
```

- sys.dm_sql_referenced_entities
  - Новое представление, пришедшее на смену **sp_depends**;
  - Возвращает одну запись для каждого объекта, на который ссылается данный объект. Например, показать все объекты, на которые ссылается хранимая процедура **MyStoredProc1**.

Приведем несколько примеров использования представления s**ys.dm_sql_referenced_entities**. В первом примере возвращаются объекты (таблицы и колонки), на которые ссылается DDL-триггер **ddlDatabaseTriggerLog**.

```
USE AdventureWorks;
GO
SELECT referenced_schema_name, referenced_entity_name, referenced_minor_name,
    referenced_minor_id, referenced_class_desc
FROM sys.dm_sql_referenced_entities ('ddlDatabaseTriggerLog',
'DATABASE_DDL_TRIGGER');
GO
```

В следующем примере показано, как получить список всех объектов, на которые ссылается пользовательская функция **dbo.ufnGetContactInformation**.

```
USE AdventureWorks;
GO
SELECT referenced_schema_name, referenced_entity_name, referenced_minor_name,
    referenced_minor_id, referenced_class_desc, is_caller_dependent, is_ambiguous
FROM sys.dm_sql_referenced_entities ('dbo.ufnGetContactInformation', 'OBJECT');
GO
```

Представление **sys.dm_sql_referenced_entities** можно также использовать для нахождения зависимостей между колонками. Ниже показан пример, в котором мы сначала создаем таблицу **Table1** с вычисляемой колонкой **c**, определенной как сумма колонок **a** и **b**. Представление **sys.dm_sql_referenced_entities** возвращает две записи — по одной для каждой из колонок, от которых зависит колонка **c**.

```
USE AdventureWorks;
GO
CREATE TABLE Table1 (a int, b int, c AS a + b);
GO
SELECT referencing_minor_name AS referencing_column,
    referenced_entity_name AS table_name,
    referenced_minor_name AS referenced_column
FROM sys.dm_sql_referenced_entities ('dbo.Table1', 'OBJECT');
GO
```

```
- Remove the table.
DROP TABLE Table1;
GO
```

```
referencing_column      table_name              referenced_column
----------------------- ----------------------- -----------------
c                       Table1                  a
c                       Table1                  b
```

В следующем примере мы удаляем таблицу **Table1** и создаем таблицу **Table2** и хранимую процедуру **Proc1**. Представление **sys.dm_sql_referenced_entities** возвращает зависимости для хранимой процедуры, указанной в качестве параметра.

```
USE AdventureWorks;
GO
IF OBJECT_ID ( 'dbo.Table1', 'U' ) IS NOT NULL
    DROP TABLE Table1;
GO
CREATE TABLE Table2 (c1 int, c2 int);
GO
CREATE PROCEDURE Proc1 AS
    SELECT a, b, c FROM Table1;
    SELECT c1, c2 FROM Table2;
GO
SELECT referenced_id, referenced_entity_name AS table_name, referenced_minor_name
AS column_name
FROM sys.dm_sql_referenced_entities ('dbo.Proc1', 'OBJECT');
GO
```

```
referenced_id, table_name,  column_name
-------------  -----------  -------------
NULL           Table1       NULL
2707154552     Table2       NULL
2707154552     Table2       c1
2707154552     Table2       c2
```

Обратите внимание на то, что для **Table1** поле **column_name** имеет значение **Null**, так как мы удалили эту таблицу перед выполнением обращения к представлению.

- sys.dm_sql_referencing_entities
  - Новое представление, пришедшее на смену **sp_depends**;
  - Возвращает одну запись для каждого объекта, который ссылается на данный объект. Например, показать все объекты, которые «пострадают» от удаления таблицы **MyTable1**.

Приведем несколько примеров использования представления **sys.dm_sql_referencing_entities**. Ниже показано, как получить список всех объектов, которые ссылаются на указанную таблицу.

```
USE AdventureWorks;
GO
SELECT referencing_schema_name, referencing_entity_name,
 referencing_id, referencing_class_desc, is_caller_dependent
FROM sys.dm_sql_referencing_entities ('Production.Product', 'OBJECT');
GO
```

Следующий пример показывает, как найти все объекты, ссылающиеся на определенный тип — **dbo.Flag** в нашем примере.

```
USE AdventureWorks;
GO
SELECT referencing_schema_name, referencing_entity_name,
 referencing_id, referencing_class_desc, is_caller_dependent
FROM sys.dm_sql_referencing_entities ('dbo.Flag', 'TYPE');
GO
```

Расширения на уровне SQL/CLR

К новинкам в SQL Server 2008 на уровне SQL/CRL отнесем возможность использования больших агрегатных значений (хранение состояния операции размером более 8 Кбайт), агрегатные функции с несколькими параметрами, возможность задания порядка сортировки и критериев уникальности при использовании табличных функций и более простой доступ к функциональности .NET Framework за счет возможности регистрации статических методов как пользовательских функций.

Расширения DDL-триггеров

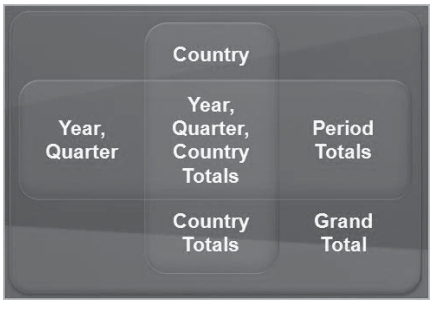
В поддержке триггеров для операций Data Definition Language (DDL) появились следующие расширения:

- Возможность включения расширенных событий для всех DDL-операций, например, использование хранимых процедур.
- Возможность использования XSD-схемы в качестве схемы для события.

Расширения для GROUP BY

Расширения для операции **GROUP BY** теперь позволяют задавать несколько группировок в рамках одной операции, что дает единый набор результатов, схожий с выполнением операции **UNION ALL** для записей, сгруппированных различными способами. Такое расширение позволяет более просто выполнять операции агрегированных запросов и создавать отчеты. Ниже приведен пример использования расширений для операции **GROUP BY** и результат его выполнения.

```
SELECT d.Year, d.Quarter, t.Country,
       SUM(f.SalesAmount) AS SalesAmount
FROM dbo.FactResellerSales AS f INNER JOIN dbo.DimDate AS d
           ON f.DateKey = d.DateKey
       INNER JOIN dbo.DimSalesTerritory AS t
           ON f.SalesTerritoryKey = t.SalesTerritoryKey
WHERE d.Year IN (2006, 2007)
GROUP BY GROUPING SETS (
    (Year, Quarter, Country),
    (Year, Quarter) ,
    (Country),
    () )
ORDER BY Year, Quarter, Country
```



Различные улучшения

В последнем разделе мы собрали ряд улучшений, появившихся в SQL Server 2008, на уровне языка Transact-SQL. К ним относятся конструкторы для табличных операций и новые операторы присваивания.

Конструкторы табличных операций позволяют вставлять в одной команде **INSERT** несколько записей, используя операцию **VALUE**:

```
INSERT INTO contacts VALUES ('John Doe', '425-333-5321'),
    ('Jane Doe', '206-123-4567'),
    ('John Smith', '650-434-7869');
```

Новые операторы присваивания — это **+=**, **-=**, ***=**, **/=**. Приведем пример их использования:

```
UPDATE Inventory
    SET quantity += s.quantity
FROM Inventory AS i INNER JOIN Sales AS s ON i.id = s.id
```

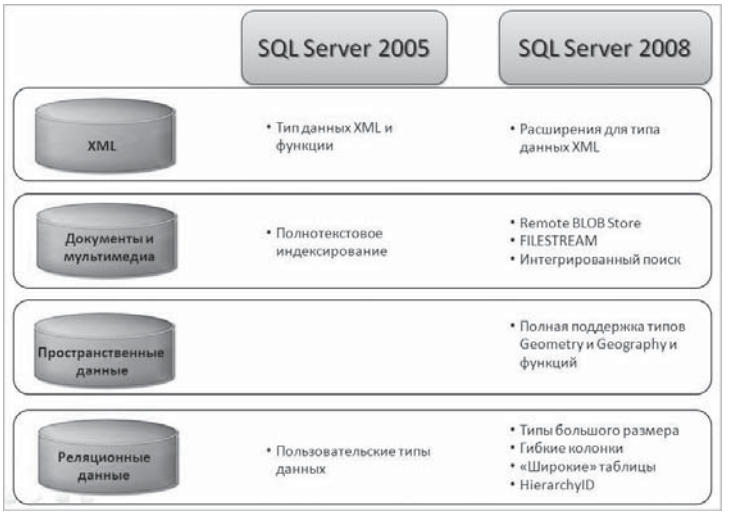
В следующем разделе мы рассмотрим основные новинки, связанные с поддержкой неструктурированных данных, появившиеся в SQL Server 2008, поддержку пространственных данных (spatial data), расширения в поддержке XML, а также ряд новых возможностей для работы с реляционными данными.

Часть 3

Расширенные типы данных

Введение

Последние годы отмечены практически лавинообразным ростом объемов данных, создаваемых в электронном виде, и, соответственно, появлением большого числа приложений для работы с такими данными. Такие данные больше не располагаются в каких-то специализированных хранилищах, они повсюду — на локальных дисках, мобильных устройствах, картах памяти и т. п. Возможность простого создания данных в электронном виде требует от приложений большей гибкости, а от систем хранения данных — поддержки типов данных, которые выходят за рамки традиционных реляционных или многомерных данных.



Документы и мультимедийные данные, XML-данные, пространственные данные — все эти типы неструктурированных, расширенных данных уже сегодня являются полноценными участниками повседневной жизни. Понимая важность полноценной поддержки новых, расширенных типов дан-

ных, Microsoft ввела в SQL Server 2008 поддержку ключевых типов на уровне ядра СУБД. Помимо этого, расширена функциональность уже поддерживаемых типов данных — XML, документов и мультимедиа, а также типов данных, задаваемых пользователями. Ключевые новинки в этой области, появившиеся в SQL Server 2008, показаны на предыдущем рисунке.

В данном разделе мы рассмотрим новые механизмы для хранения неструктурированных данных, появившиеся в SQL Server 2008, поддержку пространственных данных (spatial data), расширения в поддержке XML, а также ряд новых возможностей для работы с реляционными данными.

Хранение неструктурированных данных в SQL Server 2008

Приложения, которым требуется работа как с реляционными, так и нереляционными, неструктурированными данными, обычно используют один из следующих вариантов решения данной задачи:

- Реляционные данные хранятся в базе данных, а нереляционные бинарные данные (т. н. Binary Large Object, BLOB) хранятся в файловой системе на файловом сервере;
- Реляционные данные хранятся в базе данных, а нереляционные бинарные данные — в специальном хранилище;
- Реляционные данные хранятся в базе данных вместе с нереляционными данными.

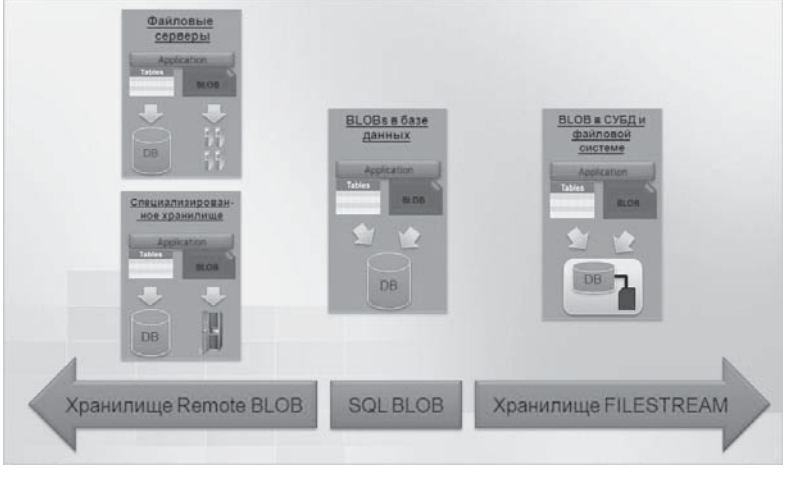
Каждый из рассмотренных вариантов имеет свои достоинства и недостатки. Например, хранение неструктурированных данных на файловых серверах или в специализированных хранилищах может существенно сократить стоимость хранения бинарных объектов, но приводит к необходимости реализации более сложных алгоритмов управления такими объектами, т. к. приложениям требуется поддержание целостности не только реляционных данных, но и данных, хранимых вне СУБД, а также синхронизация между двумя хранилищами. С другой стороны, хранение неструктурированных данных в базе данных может оказаться удобным с точки зрения обеспечения синхронизации и целостности данных, но может привести к удорожанию всего решения и, в ряде случаев, к снижению производительности.

В SQL Server 2008 появились две новые возможности для хранения неструктурированных данных — **FILESTREAM** и удаленное хранилище BLOB-объектов (**Remote BLOB Storage**).

- **FILESTREAM** — это атрибут, который устанавливается для колонки типа **varbinary**, для указания на то, что данные хранятся не в СУБД, а в файловой системе. При этом все управление данными выполняется в контексте базы данных.

- Удаленное хранилище BLOB-объектов — это набор программных интерфейсов для клиентских приложений, позволяющих эффективно управлять данными, хранящимися во внешних, по отношению к базе данных, хранилищах.

Помимо этого, в SQL Server 2008 полностью поддерживается хранение бинарных объектов в стандартных BLOB-колонках с типом данных **varbinary**. Рассмотрим каждую из перечисленных возможностей более подробно.



Варианты хранения BLOB в SQL Server 2008

Для хранения неструктурированных данных в SQL Server используются два типа данных — **binary** и введенный в SQL Server 2005 тип данных **varbinary(max)**. Тип данных **binary** позволяет хранить в колонке или переменной данного типа до 8000 байт, а **varbinary(max)** — до 2,147483647 (2^{31}) байт. Модификатор max позволяет управлять физическим расположением данных на страницах таблицы — для этого используется табличная опция **large value types out of row**. Если эта опция включена, на странице данных для конкретной записи хранится 16-байтовый указатель на присоединенные страницы, где хранятся все данные. Когда эта опция отключена, значения объемом до 8000 байт хранятся непосредственно на страницах данных, а более объемные значения — на отдельных, присоединенных страницах.

Для включения данной опции используется следующая команда:

```
sp_tableoption N'EmployeePhotos', 'large value types out of row', 'ON'
```

Новые возможности в SQL Server 2008 — FILESTREAM и Remote BLOB позволяют более эффективно решать вопросы хранения неструктурированных данных, но в тех случаях, когда размер объектов не очень большой — порядка 150-250 Кбайт, можно по-прежнему использовать тип данных **varbinary**.

FILESTREAM и хранение BLOB в файловой системе

Атрибут **FILESTREAM** может быть установлен для колонки типа **varbinary**. В этом случае SQL Server будет хранить данные из этой колонки в локальной файловой системе. Такой подход обладает двумя преимуществами:

- Производительность при использовании данных будет соответствовать потоковым возможностям файловой системы;
- Размер хранимых объектов будет ограничен объемом свободного пространства на дисковом хранилище.

Несмотря на то что данные хранятся в файловой системе, управление ими осуществляется также, как и любым другим объектом, хранимым внутри SQL Server. Описание данных как колонки с атрибутом **FILESTREAM** позволяет не только использовать средства управления и обеспечения безопасности на уровне SQL Server, но и обеспечивает целостность данных на уровне реляционных данных, хранимых в СУБД и неструктурированных данных, физически находящихся в файловой системе. Так как колонка с атрибутом **FILESTREAM** является полноценным типом данных на уровне SQL Server, на нее распространяются все операции по сопровождению, включая создание резервных копий, восстановление, интеграцию с системой безопасности и полную транзакционную поддержку.

Использование FILESTREAM — хранение BLOB во внешних файлах

Прежде чем использовать FILESTREAM, необходимо включить эту функциональность для данного экземпляра SQL Server. Для этого используется хранимая процедура **sp_filestream_configure**:

```
EXEC sp_filestream_configure
  @enable_level = 3;
```

Также можно включить поддержку **FILESTREAM** в SQL Server Management Studio — для этого используется вкладка **Advanced** диалоговой панели **Server Properties**.

После того как поддержка **FILESTREAM** включена одним из описанных выше способов, мы можем создать базу данных, использующую новые возможности SQL Server. Так как **FILESTREAM** использует специальный тип файловой группы, необходимо указать атрибут **CONTAINS FILEGROUP** для как минимум одной файловой группы для базы данных. Ниже показан пример создания базы данных с именем **FileStreamDB**. Эта база данных содержит три файловые группы — **PRIMARY**, **RowGroup1** и **FileStreamGroup1**. Группы **PRIMARY** и **RowGroup1**— это обычные файловые группы и они не могут содержать данные **FILESTREAM** — эта возможность доступна только в файловой группе **FileStreamGroup1**.

```
USE master;
GO
IF EXISTS (
  SELECT * FROM sys.databases
  WHERE name = N'FileStreamDB'
)
  DROP DATABASE FileStreamDB
GO
CREATE DATABASE FileStreamDB ON PRIMARY
  ( NAME = FileStreamDB_data,
    FILENAME = N'D:\test\FileStreamDB_data.mdf',
    SIZE = 10MB,
    MAXSIZE = 50MB,
    FILEGROWTH = 15%),
FILEGROUP RowGroup1
  ( NAME = FileStreamDB_group1,
    FILENAME = N'D:\test\FileStreamDB_group1.ndf',
    SIZE = 10MB,
    MAXSIZE = 50MB,
    FILEGROWTH = 5MB),
FILEGROUP FileStreamGroup1 CONTAINS FILESTREAM
  ( NAME = FileStreamDBResumes,
    FILENAME = N'D:\test\Resumes')
LOG ON
  ( NAME = 'FileStreamDB_log',
    FILENAME = N'D:\test\FileStreamDB_log.ldf',
    SIZE = 5MB,
    MAXSIZE = 25MB,
    FILEGROWTH = 5MB);
GO
```

Для файловой группы **FILESTREAM** атрибут FILENAME указывает каталог, в котором будут храниться файлы. Существует следующее правило: каталог вплоть до самой вложенной папки должен существовать в файло-

вой системе перед выполнением команды создания базы данных, а самая вложенная папка существовать не должна. В нашем примере — каталог **D:\test** должен существовать, а папка **Resumes** — нет.

После выполнения приведенного выше кода мы получим каталог **D:\test\Resumes** на локальном диске. В нем будут созданы два объекта — файл **filestream.hdr** и папка **$FSLOG**. Отметим, что для существующих баз данных поддержка **FILESTREAM** может быть задана добавлением соответствующей файловой группы в конструкции **ALTER DATABASE**.

После того как в базе данных появилась файловая группа с поддержкой **FILESTREAM**, можно создавать новые, или модифицировать существующие таблицы базы данных для хранения данных **FILESTREAM**. Для указания на то, что в колонке будут находиться данные FILESTREAM, необходимо создать колонку типа **varbinary(max)** и применить атрибут **FILESTREAM**. В следующем примере показано, как создать таблицу **employee**. Эта таблица содержит колонку **employee_id** с атрибутом **ROWGUIDCOL** — это требуется для использования **FILESTREAM**. Колонка Name содержит фамилию сотрудника, а колонка **Resume** — используется для хранения документа в файловой системе.

```
USE FileStreamDB;
GO
IF OBJECT_ID('dbo.employee', 'U') IS NOT NULL
  DROP TABLE dbo.employee
GO
CREATE TABLE dbo.employee
(employee_id UNIQUEIDENTIFIER ROWGUIDCOL NOT NULL UNIQUE,
  Name varchar(25),
  Resume varbinary(max) FILESTREAM);
GO
```

Для добавления данных в таблицу, использующую **FILESTREAM**, мы используем стандартную команду языка **TSQL INSERT** — с ее помощью в колонку **FILESTREAM** можно вставлять как данные типа **varbinary(max)**, так и **NULL**. В первом примере показано, как вставить **NULL** — в этом случае ядро базы данных не создаст файл в файловой системе.

```
USE FileStreamDB;
GO
INSERT INTO dbo.employee
    VALUES (newid (), 'Медведев', NULL);
```

В следующем примере показано, как вставить запись нулевой длины. Это полезно в тех случаях, когда мы планируем использовать файловые операции для манипуляции с объектами на уровне Win32 API.

```
USE FileStreamDB;
GO
INSERT INTO dbo.employee
    VALUES (newid (), 'Иванов', CAST ('' as varbinary(max)));
```

В следующем примере показано, как вставить данные в файл, находящийся в файловой системе. При выполнении команды **INSERT** произойдет преобразование данных в тип **varbinary**, создание файла и сохранение данных.

```
USE FileStreamDB;
GO
INSERT INTO dbo.employee
    VALUES (newid (), 'Миронов',
      CAST ('Миронов Сергей Иванович' as varbinary(max)));
```

Обновление данных для колонок **FILESTREAM** также выполняется средствами TSQL — это показано в следующем примере.

```
USE FileStreamDB;
GO
UPDATE dbo.employee
SET Resume = CAST ('Зайцев Юрий Петрович, 1968 г.р.' as varbinary(max))
WHERE Name = 'Зайцев';
GO
```

При удалении записи, содержащей поле **FILESTREAM**, также удаляются и соответствующие файлы в файловой системе. Для этого используется команда **DELETE**.

```
USE FileStreamDB;
GO
DELETE dbo.employee
WHERE Name = 'Соколов';
GO
```

Разработчики могут использовать данные, хранимые в колонке с атрибутом **FILESTREAM**, двумя способами. Для доступа и манипуляции с данными можно использовать конструкции языка T-SQL — точно также, как и с другими типами данных — выше мы увидели примеры использования основных команд языка TSQL.

Помимо этого, можно использовать функции Win32 API для работы с файловой системой — эти операции осуществляются в контексте транзакции SQL Server. Первый шаг — получение имени файла в формате UNC, затем — использование функции **OpenSQLFileStream()** для получения ссылки на файл. После того как ссылка получена, разработчики могут использовать стандартные функции типа **ReadFile()** и **WriteFile()** для ма-

нипуляции с данными. Так как файловые операции выполняются в рамках транзакции, удаление и переименование файлов, хранящих данные для колонок с атрибутом **FILESTREAM** на уровне файловых операций не поддерживается.

На уровне SQL Server существует функция **PathName()**, возвращающая имя файла для указанной колонки, если данные хранятся в файловой системе:

```
SELECT column_name.PathName()
```

Пример использования данной функции показан ниже.

```
DECLARE @PathName nvarchar(max)
SET @PathName = (
    SELECT TOP 1 photo.PathName()
    FROM dbo.Customer
    WHERE LastName = 'CustomerName'
    );
```

Для получения контекста транзакции используется функция **GET_FILESTREAM_TRANSACTION_CONTEXT()**:

```
SELECT GET_FILESTREAM_TRANSACTION_CONTEXT()
```

Пример использования данной функции показан ниже.

```
/* Шаг 1 - Начать транзакцию */

BEGIN TRANSACTION

/* Шаг 2 - Получить имя файла и сохранить его */

DECLARE @PathName nvarchar(max);
SET @PathName = (
    SELECT Photo.PathName()
    FROM dbo.Customer
    WHERE LastName = 'CustomerName'
    );

/* Шаг 3 - Получить контекст транзакции */

DECLARE @TransactionToken varbinary(max);
SET @TransactionToken = SELECT GET_FILESTREAM_TRANSACTION_CONTEXT ();

/* Шаг 4 - вызвать OpenSqlFilestream для получения ссылки на файл, используя имя
файла и контекст транзакции, и использовать ссылку в операциях ввода/вывода */

/* Шаг 5 - использовать функции ReadFile(), WriteFile(), TransitFile(),
SetFilePointer(), SetEndofFile() или FlushFileBuffers() */
```

```
/* Шаг 6 - Закрыть файл, используя функцию CloseHandle()*/

/* Шаг 7 - Завершить или откатить транзакцию */

COMMIT TRANSACTION
```

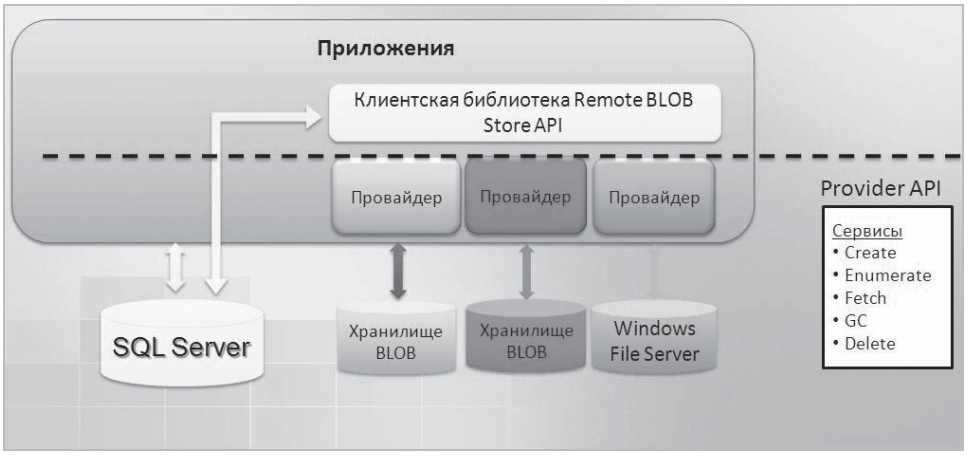
В SQL Server 2008 поддерживается только локальное хранилище (локальные диски) и поэтому ряд функций, включая «прозрачное» шифрование и использование табличных параметров не распространяется на колонки с атрибутом **FILESTREAM**. Также не поддерживается использование таких колонок при создании «слепков» баз данных (database snapshots), зеркалировании, но при этом поддерживается участие таких колонок в log shipping — оба сервера должны работать под управлением SQL Server 2008 с включенной поддержкой FILESTREAM.

Отметим, что **FILESTREAM** поддерживается и в версии SQL Server 2008 Express — ограничения на размер базы в 4 Гбайт не включают контейнер данных для **FILESTREAM**.

Remote BLOB Store API

Как мы увидели выше, использование атрибута **FILESTREAM** позволяет эффективно решать задачи, связанные с хранением неструктурированных данных в объектах файловой системы. В ряде сценариев бывает необходимо использовать не просто файловую систему, а специализированные хранилища данных, реализованные с помощью таких продуктов, как EMC Centera, Fujitsu Nearline, работающие под управлением Microsoft Windows (сервер в роли файлового хранилища) или реализованные какими-то другими способами. Использование программного интерфейса Remote BLOB Store API в SQL Server 2008 позволяет более простым способом интегрировать специализированные хранилища в решения, работающие с неструктурированными данными. Такие решения, базирующиеся на SQL Server 2008, содержат два ключевых компонента:

- **Клиентская библиотека**, позволяющая использовать любого провайдера для совместной работы с реляционными и нереляционными данными. Клиентские приложения используют эту библиотеку и любого подходящего провайдера для вставки, обновления или получения данных из специализированного хранилища вместе с данными, хранимыми в базе данных;
- **Библиотека провайдера** – компонент, обычно предоставляемый производителем специализированного хранилища, который реализует набор стандартных функций и интерфейсов для выполнения таких операций, как **Create**, **Fetch**, **Enumerate**, **Delete** и **Garbage Collect** для определенного типа хранилища.



Компоненты для работы с Remote BLOB

Программный интерфейс Remote BLOB Store API наиболее подходит для сценариев, когда требуется хранение неструктурированных данных на другом сервере и необходимы механизмы взаимодействия с другими платформами. Любая система, поддерживающая провайдера, соответствующего спецификации Remote BLOB Store Provider, или любая система, для которой может быть создан такой провайдер, может выступать в виде специализированного хранилища для неструктурированных данных для SQL Server 2008. Так как детали взаимодействия с конкретным хранилищем абстрагируются на уровне провайдера, пользователи могут заменять хранилища, не изменяя самих приложений, работающих с неструктурированными данными.

Использование Remote BLOB Store API дает разработчикам гибкость в использовании различных систем при работе с реляционными и нереляционными данными. Программный интерфейс Remote BLOB Store API поддерживает целостность между записями в базе данных и бинарными объектами, хранящимися во внешних хранилищах, через механизм связей. Например, при удалении ссылки на бинарный объект в базе данных, используя команду TSQL **DELETE**, система сама удаляет соответствующий объект из хранилища. Отметим, что несмотря на свою гибкость и удобство, Remote BLOB Store API не предоставляет разработчикам всей полноты работы с данными по сравнению с хранением бинарных объектов непосредственно в базе данных или использованием **FILESTREAM**.

После того как мы рассмотрели три способа хранения неструктурированных данных в SQL Server 2008, давайте сравним их ключевые характеристики — они перечислены в следующей таблице.

	Только файловый сервер или хранилище BLOB	BLOB в SQL Server	Remote BLOB	Колонки FILESTREAM Store API
Быстрые потоковые операции	Зависит от хранилища	Нет	Зависит от хранилища BLOB	Да
Целостные связи	Нет	Да	Да	Да
Целостность данных	Нет	Да	Нет	Да
Интегрированное управление	Нет	Да	Нет	Да
Использование удаленных серверов Windows	—	Нет	Да	В следующих версиях SQL Server
Взаимодействие с внешними хранилищами BLOB	—	Нет	Да	Нет

Таким образом, в зависимости от требований конкретного сценария, имеется возможность выбора наиболее подходящих механизмов реализации с неструктурированными данными.

Завершим наше рассмотрение работы с неструктурированными данными кратким обсуждением механизмов полнотекстового поиска. В предыдущих версиях SQL Server полнотекстовый поиск был реализован на основе внешнего сервиса — Windows Search Services, что решало основные задачи, но, в то же время, затрудняло управление, снижало производительность и вызывало ряд проблем с масштабированием. В SQL Server 2008 эти проблемы были решены за счет интеграции механизмов полнотекстового поиска непосредственно в SQL Server. Это привело к повышению производительности, особенно при использовании смешанных запросов — как показанный ниже запрос к базе данных, содержащей объекты FILESTREAM, хранящиеся в файловой системе.

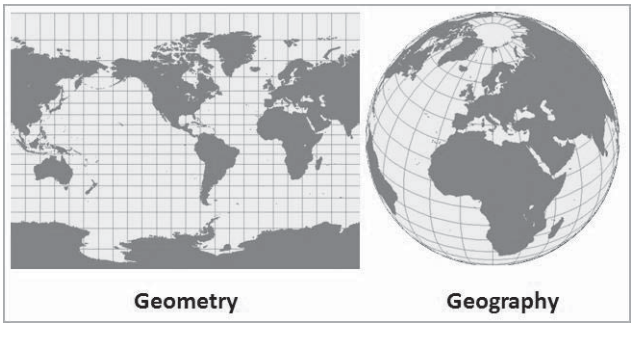
```
SELECT * FROM candidates
WHERE CONTAINS(resume,'"SQL Server"')
AND ZipCode = '123056'
```

Пространственные данные

В предыдущих версиях SQL Server поддержка пространственных данных не была реализована на уровне ядра СУБД – использование SQL Server в качестве хранилища для приложений, работающих с географическими данными было затруднено и часто требовало использования партнерских решений, расширяющих функциональность продукта.

В SQL Server 2008 появилась полноценная поддержка пространственных данных на уровне двух новых типов данных — **geometry** и **geography**, которые могут использоваться для хранения областей и точек, представляющих собой физические координаты объектов, набор новых методов для работы с такими данными (**intersects**, **buffer** и т. п.) для выполнения таких операций, как поиск пересечений областей, объединение областей и т. д., а также возможность создания индексов для пространственных данных. Пространственные данные, поддерживаемые в SQL Server 2008, полностью отвечают требованиям стандартов Open Geospatial Consortium (OGC/SQL MM, ISO 19125), интегрированы с Virtual Earth и обеспечивают скорость извлечения и обработки, сопоставимую с другими типами данных, поддерживаемыми в SQL Server.

Как мы отметили выше, в SQL Server 2008 поддерживается два типа пространственных данных — **geometry** и **geography**. Тип данных geometry реализован в SQL Server 2008 как тип данных .NET и используется для представления данных в эвклидовой (плоской) координатной системе. Тип данных **geography** также реализован как тип данных .NET и предназначен для хранения данных в эллипсоидной (пространственной) координатной системе — чаще всего к таким данным относятся данные, собираемые с GPS-приемников в виде широты и долготы.



Оба типа поддерживают более десяти типов пространственных объектов, для семи из которых можно создавать экземпляры. Эти экземпляры могут иметь свойства, которые помогают разделить их на следующие типы: **Point**, **LineString**, **Polygon**, а также группы экземпляров, собранные в коллекции **GeometryCollection**, куда могут входить такие объекты, как **MultiPoint**, **MultiLineString** и **MultiPolygon**.

Тип **Point** представляет собой объект, описывающий точку, который может иметь дополнительные значения в виде единицы измерения и возвышение (elevation). Ниже показан пример создания объекта этого типа с координатами 3, 4 и идентификатором SRID, равным 0:

```
DECLARE @g geometry;
SET @g = geometry::STGeomFromText('POINT (3 4)', 0);
```

В следующем примере показано создание объекта с координатами 3, 4, с возвышением 7 и измерением 2.5:

```
DECLARE @g geometry;
SET @g = geometry::Parse('POINT(3 4 7 2.5)');
```

И еще один пример — в нем показано, как получить значения аргументов объекта типа **Point**:

```
SELECT @g.STX;
SELECT @g.STY;
SELECT @g.Z;
SELECT @g.M;
```

Тип **MultiPoint** представляет собой коллекцию, которая может быть либо пустой, либо содержать одну или более точек. Ниже показан пример создания коллекции, содержащей две точки:

```
DECLARE @g geometry;
SET @g = geometry::STGeomFromText('MULTIPOINT((2 3), (7 8)');
```

Также для создания коллекции можно использовать метод S**TMPoint-FromText()**:

```
DECLARE @g geometry;
SET @g = geometry::STMPointFromText('MULTIPOINT((2 3), (7 8))');
```

А для получения, например, первой точки из коллекции — метод **STGeometryN()**:

```
SELECT @g.STGeometryN(1).STAsText();
```

Тип **LineString** — это одномерный объект, представляющий последовательность точек и сегментов линий, соединяющих эти точки. Экземпляр данного типа может быть либо пустым, либо содержать как минимум для точки. Приведем пример использования данного типа – на основе трех точек:

```
DECLARE @g geometry;
SET @g = geometry::STGeomFromText('LINESTRING(1 1, 2 4, 3 9)');
```

Тип **MultiLineString** представляет собой коллекцию из одного или более экземпляров **LineString**, который может быть как типа geometry, так и типа **geography**. Например:

```
DECLARE @g geometry;
SET @g = geometry::Parse('MULTILINESTRING((0 2, 1 1), (1 0, 1 1))');
```

Тип **Polygon** представляет собой 2-мерную поверхность, хранимую как последовательность точек, определяющих внешние границы, а также возможные внутренние объекты. Для задания многогранника требуется ука-

зание как минимум трех точек. Внутренние объекты не должны пересекаться. Пример задание объекта типа **Polygon** показан ниже.

```
DECLARE @g geometry;
SET @g = geometry::STPolyFromText('POLYGON((0 0, 0 3, 3 3, 3 0, 0 0), (1 1, 1 2, 2
1, 1 1))');
```

Для описания коллекции объектов типа **Polygon** используется тип **MultiPolygon**. Пример описания такого объекта приведен ниже:

```
DECLARE @g geometry;
SET @g = geometry::Parse('MULTIPOLYGON(((0 0, 0 3, 3 3, 3 0, 0 0), (1 1, 1 2, 2 1,
1 1)), ((9 9, 9 10, 10 9, 9 9)))');
SELECT @g.STGeometryN(2).STAsText();
```

И, наконец, тип **GeometryCollection** представляет собой коллекцию экземпляров типа **geometry** или **geography**. Пример использования данного типа показан ниже:

```
DECLARE @g geometry;
SET @g = geometry::STGeomCollFromText('GEOMETRYCOLLECTION(POINT
 (3 3 1), POLYGON((0 0 2, 1 10 3, 1 0 4, 0 0 2)))', 1);
```

Поддерживаемые в SQL Server 2008 типы пространственных данных имеют ряд различий, касающиеся, в основном, способов хранения данных и механизмов манипуляции ими. В планарной, плоской системе, измерения между точками и областями задаются в тех же единицах, что и координаты. Используя тип данных **geometry**, расстояние между точками (2, 2) и (5, 6) всегда будет 5, независимо от используемых единиц. В эллиптической, пространственной системе, координаты задаются указанием широты и долготы, а длины и области измеряются в метрах и квадратных метрах, хотя способ измерения может зависеть от значения Spatial Reference Identifier (SRID) для данного экземпляра типа **geography** — наиболее часто используются метры.

В планарной системе ориентация не является важным атрибутом объектов — например многогранник, описанный следующим образом:

```
((0, 0), (10, 0), (0, 20), (0, 0))
```

идентичен многограннику, описанному так:

```
((0, 0), (0, 20), (10, 0), (0, 0))
```

Стандарт OGC Simple Features не задает каких-либо правил ориентации и в SQL Server задания таких правил не требуется.

В эллиптической системе многогранник без ориентации не имеет определенного смысла. Например, один и тот же многогранник может описывать регион как в северном, так и в южном полушарии. Таким образом,

при использовании типа данных **geography** требуется указание ориентации объекта и точности описания местоположения объекта.

В SQL Server 2008 накладываются следующие ограничения при использовании типа данных **geography**:

- Каждый экземпляр типа данных **geography** должен находиться в одном полушарии. Не поддерживается хранение пространственных объектов, размер которых превышает размер полушария;
- Любой экземпляр типа **geography**, представляющий объект, больший чем полушарие, и полученный из представлений Well-Known Text (WKT) или Well-Known Binary (WKB) стандарта Open Geospatial Consortium (OGC) вызовет исключение **ArgumentException**;
- Методы типа данных geography, требующие указания двух аргументов типа **geography** — **STIntersection()**, **STUnion()**, **STDifference()** и **STSymDifference()** вернут null в тех случаях, когда результат выполнения этих методов превысит размер полушария. Это правило также относится и к методу **STBuffer()**.

Рассмотрим пример использования типа данных **geometry**. Сначала мы создаем таблицу **SpatialTable** с идентификатором id и колонкой **GeomCol1** типа **geometry**. Третья колонка — **GeomCol2** используется для представления гео-данных в формате Well-Known Text (WKT), поддерживаемом Open Geospatial Consortium (OGC)— для этого служит метод **STAsText()**:

```
CREATE TABLE SpatialTable
    (id int IDENTITY (1,1),
    GeomCol1 geometry,
    GeomCol2 AS GeomCol1.STAsText() );
GO
```

После создания таблицы мы добавляем в нее две записи. Первая запись содержит экземпляр объекта **LineString** типа **geometry**:

```
INSERT INTO SpatialTable (GeomCol1)
VALUES (geometry::STGeomFromText('LINESTRING (100 100, 20 180, 180 180)', 0));
GO
```

а вторая — экземпляр объекта **Polygon** также типа **geometry**:

```
INSERT INTO SpatialTable (GeomCol1);
VALUES (geometry::STGeomFromText('POLYGON ((0 0, 150 0, 150 150, 0 150, 0 0))', 0));
GO
```

Далее мы можем воспользоваться методами для работы с гео-данными для того, чтобы, например, найти точки пересечения двух созданных ранее объектов:

```
DECLARE @geom1 geometry;
DECLARE @geom2 geometry;
DECLARE @result geometry;
```

```
SELECT @geom1 = GeomCol1 FROM SpatialTable WHERE id = 1;
SELECT @geom2 = GeomCol1 FROM SpatialTable WHERE id = 2;
SELECT @result = @geom1.STIntersection(@geom2);
SELECT @result.STAsText();
```

Теперь рассмотрим пример создания и использования типа данных **geography**. Начнем с того, что создадим таблицу, содержащую идентификатор и колонку **GeogCol1** типа **geography**.Третья колонка – **GeomCol2** используется для представления гео-данных в формате Well-Known Text (WKT), поддерживаемом Open Geospatial Consortium (OGC)– для этого служит метод **STAsText()**:

```
CREATE TABLE SpatialTable
    ( id int IDENTITY (1,1),
    GeogCol1 geography,
    GeogCol2 AS GeogCol1.STAsText() );
GO
```

Вставим данные, описывающие объект LineString типа **geography**:

```
INSERT INTO SpatialTable (GeogCol1)
VALUES (geography::STGeomFromText('LINESTRING(47.656 -122.360, 47.656 -122.343)', 4326));
GO
```

и данные, описывающие объект Polygon:

```
INSERT INTO SpatialTable (GeogCol1)
VALUES (geography::STGeomFromText('POLYGON((47.653 -122.358, 47.649 -122.348,
47.658 -122.348, 47.658 -122.358, 47.653 -122.358))', 4326));
GO
```

Теперь воспользуемся методом **STIntersection()** для нахождения точек пересечения двух сохраненных ранее объектов:

```
DECLARE @geog1 geography;
DECLARE @geog2 geography;
DECLARE @result geography;

SELECT @geog1 = GeogCol1 FROM SpatialTable WHERE id = 1;
SELECT @geog2 = GeogCol1 FROM SpatialTable WHERE id = 2;
SELECT @result = @geog1.STIntersection(@geog2);
SELECT @result.STAsText();
```

В приведенных выше примерах мы использовали некоторые методы, поддерживаемые в SQL Server 2008 для манипуляции пространственными данными.

Методы для типа данных **geometry** показаны в следующей таблице.

STGeomFromText	Parse	STPointFromText
STMPointFromText	STLineFromText	STMLineFromText
STPolyFromText	STMPolyFromText	STGeomCollFromText
STGeomFromWKB	STPointFromWKB	STMPointFromWKB
STLineFromWKB	STMLineFromWKB	STPolyFromWKB
STMPolyFromWKB	STGeomCollFromWKB	GeomFromGml
STAsText	ToString	STAsTextZM
STAsBinary	STAsGml	STGeometryType
InstanceOf	STIsValid	MakeValid
STNumGeometries	STGeometryN	STNumPoints
STPointN	STPointOnSurface	STStartPoint
STEndpoint	STX	STY
STCentroid	STDimension	STLength
STArea	STIsEmpty.	STIsSimple
STBoundary	STEnvelope	STIsClosed
STIsRing	STExteriorRing	STNumInteriorRing
STInteriorRingN	STSrid	STEquals
STDisjoint	STIntersects	STTouches
STOverlaps	STCrosses	STWithin
STContains	STOverlaps	STRelate
STDistance	ST	BufferWithTolerance
Reduce	STConvexHull	STIntersection
STUnion	STDifference	STSymDifference
STPointOnSurface		

Методы для типа данных **geography** показаны в следующей таблице.

Методы Open Geospatial Consortium (OGC)		
STArea	STAsBinary	STAsText
STBuffer	STDimension	STDisjoint
STDistance	STEndpoint	STGeometryN
STGeometryType	STIntersection	STIntersects
STIsClosed	STIsEmpty	STLength
STNumGeometries	STNumPoints	STPointN
STSrid	STStartPoint	STUnion
Расширенные методы		
AsGml	AsTextZM	BufferWithTolerance
InstanceOf	IsNull	Lat
Long	M	NumRings
RingN	ToString	Z

Статические методы Open Geospatial Consortium (OGC)		
STGeomFromText	STPointFromText	STLineFromText
STPolyFromText	STMPointFromText	STMLineFromText
STMPolyFromText	STGeomCollFromText	STGeomFromWKB
STPointFromWKB	STLineFromWKB	STPolyFromWKB
STMPointFromWKB	STMLineFromWKB	STMPolyFromWKB

Расширенные статические методы Open Geospatial Consortium (OGC)		
GeomFromGML	Null	Parse
Point		

Завершим наше знакомство с пространственными данными кратким рассмотрением индексирования пространственных данных. Пространственный индекс — это тип расширенного индекса, который позволяет индексировать колонки с пространственными данными — колонки, содержащие типы данных **geometry** или **geography**. В SQL Server 2008 пространственные индексы строятся с использованием B-деревьев — это означает, что индексы должны представлять 2-мерные пространственные данные в линейном порядке. Таким образом, прежде чем считывать данные в пространственный индекс, SQL Server 2008 выполняет иерархическую декомпозицию, в результате которой строится четырех-уровневая табличная иерархия, в которой каждый следующий уровень декомпозирует предыдущий.

В текущей реализации пространственные индексы имеют ряд ограничений: они могут быть заданы только для таблиц, имеющих первичный ключ, максимальное число колонок с первичными ключами не должно превышать 15, максимальный размер индексного ключа не должен превышать 859 байт, пространственные ключи не могут быть указаны в индексированных представлениях, можно создать до 249 пространственных индексов для любой колонки с соответствующими данными.

Для создания пространственных индексов используется команда CREATE **SPATIAL INDEX**, для изменения индекса — команда **ALTER INDEX**, а для удаления индекса — **DROP INDEX**.

Новинки в поддержке типа данных XML

Поддержка XML впервые появилась в SQL Server 2000 — в язык TSQL были добавлены ключевые слова **FOR XML** и **OPENXML**, которые позволяли разработчикам, соответственно, извлекать результаты запросов к базам данных в виде XML-потока и сохранять XML-документы в базе данных. Эти возможности были существенно расширены в SQL Server 2005 — был введен новый тип данных xml, поддерживающий проверку на уровне XSD-схемы, выполнение XQuery-операций и индексирование. В SQL Server 2008 возможности

по работе с XML как со встроенным типом данных еще больше расширены — мы рассмотрим ключевые изменения и обновления в этом разделе.

Начнем с того, что кратко вспомним ключевые возможности по работе с XML, реализованные в предыдущих версиях SQL Server — SQL Server 2000 и SQL Server 2005. Как мы отметили выше, в SQL Server 2000 в язык TSQL были добавлены ключевые слова **FOR XML** и **OPENXML**. FOR XML — это атрибут команды **SELECT**, указывающий на то, что результаты выполнения запроса должны быть представлены в виде XML-потока. Пример использования данной функциональности показан ниже. Следующий запрос:

```
SELECT ProductID, ProductName
FROM Products Product
FOR XML AUTO
```

вернет следующий XML-документ:

```
<Product ProductID="1" ProductName="Widget"/>
<Product ProductID="2" ProductName="Sprocket"/>
```

Функция **OPENXML** предназначена для выполнения обратных действий — создания записи на основе переданного ей XML-документа. Пример использования данной функциональности показан ниже. Следующий запрос:

```
DECLARE @doc nvarchar(1000)
SET @doc = '<Order OrderID = "1011">
          <Item ProductID="1" Quantity="2"/>
          <Item ProductID="2" Quantity="1"/>
      </Order>'
DECLARE @xmlDoc integer
EXEC sp_xml_preparedocument @xmlDoc OUTPUT, @doc
SELECT * FROM
OPENXML (@xmlDoc, 'Order/Item', 1)
WITH
(OrderID integer '../@OrderID',
 ProductID integer,
 Quantity integer)
EXEC sp_xml_removedocument @xmlDoc
```

приведет к созданию такой записи:

```
OrderID   ProductID    Quantity

1011      1            2

1011      2            1
```

Обратите внимание на использование хранимых процедур **sp_xml_preparedocument** и **sp_xml_removedocument** для создания XML-документа в памяти и его удаления после записи в базу данных.

В SQL Server 2005 атрибут **FOR XML** был расширен возможностью задания новых опций для корневых элементов и имен элементов документа, была включена поддержка вложенных вызовов запросов с **FOR XML**, позволяющих создавать сложные иерархии внутри XML-документов, а также новый режим **PATH**, позволяющий описать структуру получаемого XML-документа с помощью синтаксиса XPath. Пример использования данной функциональности показан ниже. Следующий запрос:

```
SELECT ProductID AS '@ProductID',
   ProductName AS 'ProductName'
FROM Products
FOR XML PATH ('Product'), ROOT ('Products')
```

создаст такой XML-документ:

```
<Products>
<Product ProductID="1">
        <ProductName>Widget</ProductName>
   </Product>
<Product ProductID="2">
        <ProductName>Sprocket</ProductName>
   </Product>
</Products>
```

Помимо расширений функциональности, впервые появившейся в SQL Server 2000, в SQL Server 2005 появился встроенный тип данных **xml**, использование которого позволяет создавать переменные и колонки для хранения XML-данных. Пример использования данной функциональности показан ниже.

```
CREATE TABLE SalesOrders
(OrderID integer PRIMARY KEY,
 OrderDate datetime,
 CustomerID integer,
 OrderNotes xml)
```

Тип данных **xml** может использоваться для хранения в базе данных отформатированных документов (HTML, XML, XHTML и т. п.) или полуструктурированных данных. Можно хранить нетипизованные или типизованные XML-данные — последние могут быть проверены на соответствие XSD-схеме. Для задания схемы, используемой для проверки вводимых данных, используется команда **CREATE XML SCHEMA COLLECTION**:

```
CREATE XML SCHEMA COLLECTION ProductSchema AS
'<?xml version="1.0" encoding="UTF-16"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <!- Здесь располагается сама схема ->
</xs:schema>'
```

После того как схема описана, она ассоциируется с xml-переменной или колонкой соответствующего типа — пример показан ниже.

```
CREATE TABLE SalesOrders
(OrderID integer PRIMARY KEY,
 OrderDate datetime,
 CustomerID integer,
 OrderNotes xml(ProductSchema))
```

Проверка типизованного XML на соответствие ассоциированной с переменной или колонкой соответствующего типа схеме происходит при записи или обновлении данных — эта возможность позволяет, например, гарантировать соответствие вводимых данных принятым стандартам или обеспечить совместимость с документами различных типов.

Тип данных **xml** также поддерживает ряд методов, которые могут использоваться для выполнения запросов или манипуляции с XML-данными. Например, можно использовать метод **query** для выборки данных, как это показано в следующем примере.

```
declare @x xml
set @x=
'<Invoices>
<Invoice>
      <Customer>Kim Abercrombie</Customer>
      <Items>
         <Item ProductID="2" Price="1.99" Quantity="1" />
         <Item ProductID="3" Price="2.99" Quantity="2" />
         <Item ProductID="5" Price="1.99" Quantity="1" />
      </Items>
</Invoice>
   <Invoice>
      <Customer>Margaret Smith</Customer>
      <Items>
         <Item ProductID="2" Price="1.99" Quantity="1"/>
      </Items>
   </Invoice>
</Invoices>'
SELECT @x.query(
'<CustomerList>
{
for $invoice in /Invoices/Invoice
return $invoice/Customer
}
</CustomerList>')
```

В приведенном выше запросе используется XQuery-синтаксис, с помощью которого ищутся все элементы **Invoice**, находящиеся в данном документе, и возвращается XML-документ, который содержит элемент **Customer** для каждого элемента **Invoice** — пример результирующего документа показан ниже.

```
<CustomerList>
  <Customer>Kim Abercrombie</Customer>
  <Customer>Margaret Smith</Customer>
</CustomerList>
```

Другая новинка, относящаяся к поддержке XML и появившаяся в SQL Server 2005 — это поддержка XML-индексов. Имеется возможность создания первичных и вторичных XML-индексов для колонок типа **xml**, что позволяет повысить производительность запросов к XML-данным. Первичный XML-индекс — это сжатое представление все ветвей XML-документа, которые процессор обработки запросов использует для быстрого нахождения ветвей в документе. После того как первичный XML-индекс создан, можно создать вторичный XML-индекс, который поможет повысить производительность при выполнении ряда специфических запросов. В следующем примере показано, как создать первичный XML-индекс и вторичный XML-индекс типа PATH, который позволит улучшить производительность при выполнении XPath- запросов к данному XML-документу.

```
CREATE PRIMARY XML INDEX idx_xml_Notes
ON SalesOrders (Notes)
GO

CREATE XML INDEX idx_xml_Path_Notes
ON SalesOrders (Notes)
USING XML INDEX idx_xml_Notes
FOR PATH
GO
```

Функциональность, реализованная в SQL Server 2000 и 2005, была расширена в SQL Server 2008. К ключевым расширениям в области поддержки работы с XML в SQL Server 2008 можно отнести улучшенные возможности проверки данных на соответствие схеме, расширенную поддержку XQuery и расширенную функциональность при вставке XML-данных средствами DML (Data Manipulation Language).

Расширения XSD

Проверка данных на соответствие схеме позволяет убедиться в том, что XML-документ, хранимый в SQL Server, соответствует определенному стандарту и заданным на уровне схемы бизнес-правилам. На уровне схемы задаются допустимые в XML-документе элементы и атрибуты, что позво-

ляет убедиться в том, что XML-документ содержит требуемые данные в рамках предопределенной структуры.

В SQL Server 2005 появилась поддержка проверки XML-данных на основе коллекций XSD- схем. Подход заключается в том, что вы создаете коллекцию схем, которая содержит схемы с правилами для XML-данных, используя команду **CREATE XML SCHEMA COLLECTION**, а затем ссылаетесь на имя коллекции при задании колонки или переменной типа **xml**, которая должна соответствовать правилам, описанным на уровне схемы. SQL Server выполняет проверку вводимых или обновляемых данных на соответствие указанной коллекции схем. В SQL Server 2005 реализовано подмножество полной спецификации XML Schema и поддерживаются ключевые сценарии проверки вводимых XML-данных.

В SQL Server 2008 поддержка XSD-схем расширена за счет введения дополнительных возможностей, к которым относятся поддержка проверки на уровне **any** (т. н. **lax validation**), полная поддержка проверки на уровне **dateTime**, **time** и **date**, включая сохранение информации о часовых поясах и улучшенная поддержка типов **union** и **list**.

Поддержка проверки на уровне шаблонов реализована на уровне конструкций **any**, **anyAttribute** и **anyType**. Например, следующая схема:

```
<xs:complexType name="Order" mixed="true">
  <xs:sequence>
    <xs:element name="CustomerName"/>
    <xs:element name="OrderTotal"/>
    <xs:any namespace="##other" processContents="skip"
minOccurs="0" maxOccurs="unbounded"/>
  </xs:sequence>
</xs:complexType>
```

задает XML-элемент с именем **Order**, который должен содержать вложенные элементы с именами **CustomerName** и **OrderTotal**. Помимо этого, элемент может содержать неограниченное число других элементов, относящихся к пространствам имен, отличным от того, в котором определен тип **Order**. Следующий XML-документ содержит экземпляр элемента **Order**, соответствующий описанию схемы. Обратите внимание на то, что в документе также содержится элемент **shp:Delivery**, который не описан в схеме.

```
<Invoice xmlns="http://adventure-works.com/order"
   xmlns:shp="http://adventure-works.com/shipping">
  <Order>
    <CustomerName>Graeme Malcolm</CustomerName>
    <OrderTotal>299.99</OrderTotal>
    <shp:Delivery>Express</shp:Delivery>
  </Order>
</Invoice>
```

Проверка на соответствие шаблону зависит от атрибута **processContents** для секции схемы, в которой описываются шаблоны. В SQL Server 2005 схемы могут содержать значения атрибута **processContents** — **skip** и **strict** для объявлений **any** и **anyAttribute**. В предыдущем примере атрибут **processContents** для шаблона имел значение **skip** — таким образом, содержимое данного элемента не проверялось. Даже если в схеме описан элемент **shp:Delivery**, он не будет проверяться до тех пор, пока в описании шаблона для элемента **Order** значение атрибута **processContents** не будет установлено в **strict**.

В SQL Server 2008 добавлено третье возможное значение атрибута **processContents — lax**, которое позволяет указать на то, что все элементы, описанные в схеме, должны быть включены в проверку, а элементы, не содержащиеся в схеме, могут быть проигнорированы. Таким образом, если в предыдущем примере присвоить атрибуту **processContents** для шаблона значение **lax**, и добавить в схему описание элемента **shp:Delivery**, этот элемент будет включен в проверку. Но так как элемент **shp:Delivery** не описан в схеме, он не будет включен в проверку. Помимо этого, спецификация XML Schema определяет, что атрибут **anyType** автоматически подлежит проверке согласно описанным выше правилам. В SQL Server 2005 lax-проверка не поддерживалась — по умолчанию выполнялась проверка на уровне **strict**.

Для задания данных, описывающих дату и время, в XML-схемах используется тип данных **dateTime**. Дата и время задаются в следующем формате: **2007-12-01T21:11:20:000Z**, который представляет собой 1-е декабря 2007 года, 11 часов 20 минут по Гринвичу — UTC (**000Z**). Другие часовые пояса в следующем формате: **000+3:00**, например, описывает московское время. Спецификация XML Schema задает компонент часового пояса типов данных **dateTime**, **date** и **time** как опциональный. Тем не менее, в SQL Server 2005 требовалось указание часового пояса при задании данных типа **date-Time**, **date** и **time**. Помимо этого, в SQL Server 2005 данные о часовом поясе не сохранялись, а приводились к UTC — например, значение **2007-12-25T06:00:00:000-8:00** превращалось в **2007-12-25T14:00:00:000Z**. В SQL Server 2008 эти ограничения удалены — при задании даты и времени можно не указывать часовой пояс, но если он указан, данные сохраняются корректно.

Разработчики могут использовать XML-схемы для задания типов данных для XML-данных так, что эти данные могут содержать набор значений, присваиваемых элементам и атрибутам. Например, можно определить тип **sizeListType**, который ограничивает список возможных значений, присваиваемых элементу **AvaliableSizes** до **S**, **M** и **L**. В SQL Server 2005 поддерживаются схемы, содержащие простые определения типов и соответствующие ограничения. Например, можно использовать тип **list** для задания возможных размеров, как показано в следующем примере:

```
<xs:simpleType name="sizeListType">
  <xs:list>
    <xs:simpleType>
      <xs:restriction base="xs:string">
        <xs:enumeration value="S"/>
        <xs:enumeration value="M"/>
        <xs:enumeration value="L"/>
      </xs:restriction>
    </xs:simpleType>
  </xs:list>
</xs:simpleType>
```

Такое описание схемы позволяет создать элемент, который содержит все возможные размеры в виде списка значений, разделенных пробелами – это показано в следующем примере:

```
<AvailableSizes>S M L</AvailableSizes>
```

Предположим, что нам нужно предоставить два различных способа описания возможных размеров продукта. Предположим, например, что одежда имеет размеры 46, 48 и 50, но также может описываться как **S**, **M** и **L**. Для поддержки такой возможности в SQL Server 2008 введен тип **union**, который содержит несколько типов **list**, которые могут объединяться в единый тип, описывающий возможные значения для определенного типа. В приведенном ниже примере показано, как создать XML-схему, содержащую тип **productSizeType**, для которого допустимыми значениями будут числовые значения (46, 48, 50) и символьные значения (S, M L).

```
CREATE XML SCHEMA COLLECTION CatalogSizeSchema AS
N'<?xml version="1.0" encoding="UTF-16"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
   <xs:simpleType name="productSizeType">
      <xs:union>
         <xs:simpleType>
           <xs:list>
            <xs:simpleType>
              <xs:restriction base="xs:integer">
               <xs:enumeration value="46"/>
               <xs:enumeration value="48"/>
               <xs:enumeration value="50"/>
              </xs:restriction>
            </xs:simpleType>
           </xs:list>
         </xs:simpleType>
         <xs:simpleType>
           <xs:list>
```

```
            <xs:simpleType>
              <xs:restriction base="xs:string">
                <xs:enumeration value="S"/>
                <xs:enumeration value="M"/>
                <xs:enumeration value="L"/>
              </xs:restriction>
            </xs:simpleType>
          </xs:list>
        </xs:simpleType>
      </xs:union>
   </xs:simpleType>
</xs:schema>'
```

При таком описании схемы элементы, базирующиеся на типе **productSizeType**, могут содержать значения из любого списка — в следующем примере показано, что оба элемента будут соответствовать правилам, заданным в схеме.

```
<Catalog>
  <Product>
    <ProductName>Polo Shirt</ProductName>
    <AvailableSizes>46 48</AvailableSizes>
  </Product>
  <Product>
    <ProductName>Long-Sleeve Shirt</ProductName>
    <AvailableSizes>S M L</AvailableSizes>
  </Product>
</Catalog>
```

Точно также в SQL Server 2008 поддерживаются задания в схеме типов **list**, содержащих внутри себя типы **union**.

Расширения XQuery

Как мы отметили выше, поддержка типа данных xml впервые появилась в SQL Server 2005. Эта поддержка, в частности, включала предоставление разработчикам ряда методов для манипуляции XML-данными, хранимыми в переменных или колонках соответствующего типа. Большинство операций над XML-данными можно выполнить, используя синтаксис **XQuery**, который поддерживает как навигацию по данным, так и манипуляции с этими данными. Синтаксис **XQuery**, поддерживаемый в SQL Server 2005, включает такие конструкции, как **for**, **where**, **order by** и **return**, которые вместе называются FLWOR-выражениями и могут использоваться для перебора ветвей XML-документа и возврата найденных значений. В SQL Server 2008 добавлена конструкция **let**, позволяющая присваивать значения переменным в выражениях **XQuery** — пример использования данной конструкции показан ниже.

```
declare @x xml
set @x=
'<Invoices>
<Invoice>
      <Customer>Kim Abercrombie</Customer>
      <Items>
         <Item ProductID="2" Price="1.99" Quantity="1" />
         <Item ProductID="3" Price="2.99" Quantity="2" />
         <Item ProductID="5" Price="1.99" Quantity="1" />
      </Items>
</Invoice>
   <Invoice>
      <Customer>Margaret Smith</Customer>
      <Items>
         <Item ProductID="2" Price="1.99" Quantity="1"/>
      </Items>
   </Invoice>
</Invoices>'
SELECT @x.query(
'<Orders>
{
for $invoice in /Invoices/Invoice
let $count :=count($invoice/Items/Item)
order by $count
return
<Order>
{$invoice/Customer}
<ItemCount>{$count}</ItemCount>
</Order>
}
</Orders>')
```

В результате выполнения данного кода будет создан следующий XML-документ:

```
<Orders>
   <Order>
      <Customer>Margaret Smith</Customer>
      <ItemCount>1</ItemCount>
   </Order>
   <Order>
      <Customer>Kim Abercrombie</Customer>
      <ItemCount>3</ItemCount>
   </Order>
</Orders>
```

Обратите внимание на то, что в SQL Server 2008 не поддерживается присваивание значений элементам, создаваемым на лету.

Расширения XML DML

Помимо рассмотренной выше поддержки выражений XQuery для выполнения манипуляций над XML-данными, тип данных xml поддерживает такие XML DML команды, как **insert**, **replace value of** и **delete** – для этого используется метод **modify**. Эти команды могут использоваться для манипуляции XML-данными, хранимыми в переменных или в колонках типа **xml**. В SQL Server 2008 также поддерживается использование переменных типа **xml** при выполнении команды **insert** для вставки XML-данных в существующую XML-структуру. Например, предположим, что переменная **@productList** типа **xml** содержит следующий документ:

```
<Products>
  <Bike>Mountain Bike</Bike>
  <Bike>Road Bike</Bike>
</Products>
```

Мы можем использовать следующий код для вставки нового продукта в уже существующий документ:

```
DECLARE @newBike xml
SET @newBike = '<Bike>Racing Bike</Bike>'
@productList.modify
('insert sql:variable("@newBike") as last into (/Products)[1]')
```

После выполнения этого кода переменная **@productList** будет содержать следующий XML-документ:

```
<Products>
  <Bike>Mountain Bike</Bike>
  <Bike>Road Bike</Bike>
  <Bike>Racing Bike</Bike>
</Products>
```

Расширения для хранения реляционных данных

В завершение кратко рассмотрим ряд новинок, появившихся в области поддержки работы с реляционными данными. К ним относятся:

- **HierarchyID** — возможность хранения иерархических данных;
- **Large UDT** — возможность создания пользовательских типов (User Defined Types, UDT) размером более 8 Кбайт;

- **Sparse Columns** — оптимизированное хранение колонок с «разреженными» данными;
- **Wide Tables** — поддержка сотен тысяч колонок с «разреженными» данными;
- **Filtered Indices** — возможность задания индексов для подмножества данных в таблицах.

Возможность хранения иерархических данных реализована в SQL Server 2008 на основе системного типа данных **hierarchyid**. Этот тип следует использовать при создании таблиц с иерархической структурой. Набор методов, включенных в состав TSQL, позволяет выполнять запросы и манипулировать иерархическими данными.

Под иерархическими данными понимается набор данных, элементы которого взаимосвязаны в виде каких-либо структур. Иерархические взаимосвязи — это когда один элемент является родительским по отношению к другому элементу. Примерами иерархических данных могут быть:

- Структура организации;
- Файловая система;
- Набор задач в рамках проекта;
- Ссылки между веб-страницами;
- и т. п.

Значение типа **hierarchyid** описывает местоположение элемента в древовидной структуре. Эти значения обладают следующими характеристиками: компактность — описание элемента в организационной структуре, описывающей 100 000 сотрудников с 6-ю уровнями занимает 5 байт, эффективное сравнение элементов и поддержка вставки и удаления элементов в любой точке иерархии.

К ограничениям данного типа отнесем тот факт, что данные, находящиеся в колонке типа **hierarchyid** не обязательно представляет собой дерево — заполнение колонки корректными данными — это задача приложения. Помимо этого, в ряде сценариев можно использовать элементы типа **hierarchyid** для хранения ссылок на местоположения в иерархиях, описанных в других таблицах. Использование типа **hierarchyid** не гарантирует уникальности значений — приложения должны самостоятельно проверять сохраняемые в колонках значения, либо использовать механизмы уникальных ключей. Приложения должны самостоятельно отслеживать корректность взаимоотношений между элементами, особенно при удалении элементов — например, при удалении родительского элемента также должны быть удалены и дочерние элементы.

На уровне языка TSQL и типа данных **SqlHierarchyId**, доступного в коде на управляемых языках программирования (C# и Visual Basic .NET) через пространство имен **Microsoft.SqlServer.Types.SqlHierarchyId**, поддерживаются следующие методы для работы с типом данных **hierarchyid**:

Метод	Описание
GetAncestor()	Возвращает значение типа **hierarchyid**, представляющее родительский элемент указанного уровня
GetDescendant()	Возвращает значение типа **hierarchyid**, представляющее дочерний элемент
GetLevel()	Возвращает целочисленное значение, представляющее собой глубину элемента в иерархии
GetRoot()	Возвращает корневой элемент иерархии
IsDescendant()	Позволяет узнать, является ли указанный элемент дочерним элементом
Parse()	Преобразует строчное значение идентификатора иерархии в тип данных **hierarchyid**. Обратное преобразование выполняется с помощью метода **ToString()**
Read()	Используется только в .NET-коде для получения данных из **BinaryReader** и присвоения значения типа **SqlHierarchyId** соответствующему объекту. В TSQL для этой операции следует использовать функции **CAST** и **CONVERT**
Reparent()	Позволяет изменить родительский элемент для указанного элемента
ToString()	Преобразует тип данных **hierarchyid** в строчное значение. Обратное преобразование выполняется с помощью метода **Parse()**
Write()	Используется только в .NET-коде для записи типа данных **hierarchyid** в **BinaryReader**. В TSQL для этой операции следует использовать функции **CAST** и **CONVERT**

Рассмотрим небольшой пример использования типа данных **hierarchyid**. Обратимся к базе данных AdventureWorks, которая поставляется вместе с SQL Server 2008. Таблица **Employee** представляет собой данные о сотрудниках компании и через поле **ManagerID** описывает иерархические взаимоотношения. Выполнение следующего запроса:

```
SELECT
     Mgr.EmployeeID AS MgrID, Mgr.LoginID AS Manager,
     Emp.EmployeeID AS E_ID, Emp.LoginID, Emp.Title
FROM HumanResources.Employee AS Emp
LEFT JOIN HumanResources.Employee AS Mgr
ON Emp.ManagerID = Mgr.EmployeeID
ORDER BY MgrID, E_ID
```

позволяет получить представление о том, как организованы данные в этой таблице:

```
MgrID Manager                E_ID LoginID                 Title
NULL NULL                     109 adventure-works\ken0    Chief Executive Officer
3    adventure-works\roberto0 4   adventure-works\rob0    Senior Tool Designer
3    adventure-works\roberto0 9   adventure-works\gail0   Design Engineer
3    adventure-works\roberto0 11  adventure-works\jossef0 Design Engineer
3    adventure-works\roberto0 158 adventure-works\dylan0  Research and Development Manager
3    adventure-works\roberto0 263 adventure-works\ovidiu0 Senior Tool Designer
3    adventure-works\roberto0 267 adventure-works\michael8 Senior Design Engineer
3    adventure-works\roberto0 270 adventure-works\sharon0 Design Engineer
6    adventure-works\david0   2   adventure-works\kevin0  Marketing Assistant
...
```

Обратим внимание на то, что менеджеру с идентификатором **MgrID** подчиняются семь сотрудников. Путем создания более сложных запросов можно получить детальную информацию об этих сотрудниках, но, так как мы рассматриваем использование типа данных **hierarchyid**, давайте создадим новую таблицу, которая позволит нам более просто описать данную иерархию.

Создадим новую таблицу с колонкой **OrgNode** типа **hierarchyid**. Данная колонка заменит колонки **EmployeeID** и **ManagerID** в существующей таблице. Колонка **OrgNode** будет являться первичным ключом и, таким образом, нам гарантировано наличие в ней уникальных значений. Кластерный индекс для колонки **OrgNode** будет содержать соответствующие данные. Выполним следующий TSQL-код:

```
USE AdventureWorks;
GO

CREATE TABLE HumanResources.NewOrg
(
  OrgNode hierarchyid,
  EmployeeId int,
  LoginID nvarchar(50),
  ManagerID int,
  Title nvarchar(100),
  HireDate datetime
CONSTRAINT PK_NewOrg_OrgNode
  PRIMARY KEY CLUSTERED (OrgNode)
)
GO
```

Затем создадим временную таблицу **#Children** с колонкой **Num**, в которой будет сохранено число дочерних элементов для каждого элемента:

```
CREATE TABLE #Children

   (
    EmployeeId int,
    ManagerID_int,
____Num_int
);
GO
```

Для ускорения выполнения запроса при заполнении таблицы **NewOrg** создадим кластерный индекс:

```
CREATE CLUSTERED INDEX tmpind ON #Children(ManagerID, EmployeeID);
```

Заполним таблицу **#Children**, используя метод **ROW_NUMBER()**, который позволит нам обойти ограничение на использование вложенных запросов с агрегатами в рекурсивных запросах:

```
INSERT #Children (EmployeeID, ManagerID, Num)
SELECT EmployeeID, ManagerID,
  ROW_NUMBER() OVER (PARTITION BY ManagerID ORDER BY ManagerID)
FROM HumanResources.EmployeeDemo ;
GO
```

Выполним следующий запрос:

```
SELECT * FROM #Children ORDER BY ManagerID, Num;

GO
```

и обратим внимание на то, что теперь колонка **Num** содержит последовательность чисел для каждого **ManagerID**:

```
EmployeeID ManagerID Num
---------- --------- ---
109        NULL      1
4          3         1
9          3         2
11         3         3
158        3         4
...
271        6         1
272        6         2
...
```

Теперь заполним таблицу **HumanResources.NewOrg**, используя методы **GetRoot** и **ToString** для объединения значений из колонки **Num** в иерархический формат, а затем обновим значение колонки **OrgNode**, записав в нее результирующие значения:

```
WITH paths(path, EmployeeID)
AS (
– This section provides the value for the root of the hierarchy
SELECT hierarchyid::GetRoot() AS OrgNode, EmployeeID
FROM #Children AS C
WHERE ManagerID IS NULL

UNION ALL
– This section provides values for all nodes except the root
SELECT
CAST(p.path.ToString() + CAST(C.Num AS varchar(30)) + '/' AS hierarchyid),
C.EmployeeID
FROM #Children AS C
JOIN paths AS p
   ON C.ManagerID = P.EmployeeID
)
INSERT HumanResources.NewOrg (OrgNode, O.EmployeeID, O.LoginID, O.ManagerID,
O.Title, O.HireDate)
SELECT P.path, O.EmployeeID, O.LoginID, O.ManagerID, O.Title, O.HireDate
FROM HumanResources.EmployeeDemo AS O
JOIN Paths AS P
```

```
   ON O.EmployeeID = P.EmployeeID;
GO
```

Значения, хранимые в колонке типа **hierarchyid**, будут более понятными, если перевести их в символьный формат. Выполним следующий код для получения представления о том, как хранятся описания иерархических данных:

```
SELECT OrgNode.ToString() AS LogicalNode, *
FROM HumanResources.NewOrg
ORDER BY LogicalNode;
GO
```

Завершим наше упражнение, удалив временную таблицу:

```
DROP TABLE #Children;
GO
```

Как мы отметили выше, в SQL Server 2008 поддерживается оптимизированное хранение колонок с «разреженными» данными. Такие данные обычно появляются при реализации сценариев, требующих, например, хранения объектов в виде сериализованных данных в базе данных, хранении каталогов продуктов с большим числом атрибутов, при создании систем управления документами, систем для хранения гео-данных и т. п. Одним словом, любая попытка хранения данных, описывающих объекты, у которых может быть большое число атрибутов, приводит к появлению полуструктурированных данных, часть из которых может иметь пустые значения. Пример «разреженных» данных показан ниже.

pk	c1	sc1	sc2	sc3	sc4	sc5	sc6	sc7	sc8	sc9
1	A	1								9
2	B		2		4					
3	C						6	7		
4	D	1				5				
5	E				4				8	
6	F			3						9
7	G					5		7		
8	H		2						8	
9	I			3			6			

В SQL Server 2008 такие данных хранятся с применением ряда алгоритмов оптимизации, которые позволяют, например, не тратить место на пустые значения, а под ненулевые значения (показанные выше) отводить от 2 до 4 байт:

pk	c1	sc1	sc2	sc3	sc4	sc5	sc6	sc7	sc8	sc9
1	A	(sc1, sc9) (1,9)								
2	B	(sc2, sc4) (2,4)								
3	C	(sc6, sc7) (6, 7)								
4	D	(sc1, sc5) (1,5)								
5	E	(sc4, sc8) (4, 8)								
6	F	(sc3, sc9) (3,9)								
7	G	(sc5, sc7) (5, 7)								
8	H	(sc2, sc8) (2, 8)								
9	I	(sc3, sc6) (3, 6)								

Такой подход обладает существенным преимуществом — в одной таблице можно хранить сотни тысяч колонок с разреженными данными (т. н. Wide Tables), но доступ к таким данным будет чуть медленнее по сравнению с доступом к «обычным» данным, так как ядру базы данных потребуется время на преобразование оптимизированного представления данных в «разреженное».

Также поддерживаемый в SQL Server 2008 «набор колонок» (column set) позволяет использовать «разреженные» колонки как группу при выполнении операций извлечения и обновления данных. Набор колонок представляет собой логическую группу «разреженных» колонок в таблице. Например:

```
Create Table Products(Id int, Type nvarchar(16),
ProductProperties XML COLUMN_SET
FOR ALL_SPARSE_COLUMNS);
```

В данном примере мы получаем обновляемую, вычисляемую XML-колонку, для которой операция **SELECT *** возвращает все не «разреженные» колонки, а также набор «разреженных» колонок. Таким образом, мы можем работать со всеми «разреженными» колонками как с единым набором.

И, наконец, использование **Filtered Indices** в SQL Server 2008 предоставляет нам возможность задания индексов для подмножества данных в таблицах. Среди сценариев использования таких индексов отметим следующие: индексирование свойств, специфичных для определенного типа данных, индексирование активных данных, индексирование данных в определенных разделах. Например, чтобы создать индекс для набора данных C1 (см. предыдущий пример), можно выполнить следующий код:

```
Create index i1 on T(sc1) where C1=A or C1=D;
Select sc1 from T where c1=A and sc1>5
```

pk	c1	sc1	sc2	sc3	sc4	sc5	sc6	sc7	sc8	sc9
1	A	1								9
2	B		2		4					
3	C						6	7		
4	D	1				5				
5	E				4				8	
6	F			3						9
7	G					5		7		
8	H		2						8	
9	I			3			6			

Еще один пример создания индексов для «разреженных» колонок:

```
Create index i1 on T(sc1) where C1=A or C1=D;
Select sc1 from T where c1=A and sc1>5
Create index i2 on T(sc7) where sc7 IS NOT NULL;
Select sc1 from T where sc7=5
```

И результат выполнения данного кода:

pk	c1	sc1	sc2	sc3	sc4	sc5	sc6	sc7	sc8	sc9
1	A	1								9
2	B		2		4					
3	C						6	7		
4	D	1				5				
5	E				4				8	
6	F			3						9
7	G					5		7		
8	H		2						8	
9	I			3			6			

Заключение

В данном разделе мы рассмотрели ключевые новинки в SQL Server 2008, связанные с поддержкой расширенных типов данных, пространственных данных (spatial data), расширения в поддержке типа данных XML, а также ряд новых возможностей для работы с реляционными данными.

В следующем разделе мы рассмотрим ряд технологий, облегчающих создание приложений на основе SQL Server 2008. К таким технологиям относятся поддержка использования бизнес-сущностей (Entity Framework), интегрированный язык запросов (LINQ) и возможность синхронизации данных от любых источников (Microsoft Sync Framework).

Глава 4

Динамичная разработка

Введение

В данном разделе мы рассмотрим ряд технологий, непосредственно связанных с Microsoft SQL Server 2008, и позволяющих более эффективно создавать различные классы приложений, использующие данный сервер. К таким технологиям относятся: программная библиотека для поддержки работы с данными на уровне бизнес-сущностей **ADO.NET Entity Framework**, поддержка возможности задания запросов к базе данных непосредственно в языках программирования C# и Visual Basic .NET — **Language Integrated Query** (LINQ), а также механизм синхронизации данных **Microsoft Sync Framework** (MSF). Эти технологии входят в группу, известную под названием **Microsoft Data Programmability**. Задача этих технологий, совместно с Microsoft SQL Server 2008, состоит в решении ключевой задачи, которая возникла из-за того, что системы управления и языки программирования долгие годы развивались самостоятельно, концентрируясь на расширении специфической для них функциональности и на сегодняшний день функциональность хранилищ данных требует синхронизации с возможностями языков программирования.

Как мы увидели в предыдущих разделах, платформа для работы с данными компании Microsoft, куда ключевым компонентом входит SQL Server 2008, может быть охарактеризована как продуктивная, всеобъемлющая и масштабируемая. Microsoft SQL Server 2008 совместно со средством разработки Visual Studio позволяет разработчикам достигать больших результатов за счет использования новых моделей данных, синтаксиса и новых средств командной разработки. Microsoft SQL Server 2008 поддерживает как новые, так и существующие технологии связи с источниками данных, что позволяет разработчикам использовать наиболее подходящие для конкретного проекта сценарии связи с данными. И, наконец, масштабируемость SQL Server 2008 позволяет использовать этот продукт для реализации различных сценариев — от работы на мобильных устройствах до корпоративных приложений.

Рассмотрим каждую из перечисленных характеристик более подробно. Начнем с *продуктивности*. Разработчикам часто приходится проводить много времени за изучением схем баз данных и написанием комплексных запросов для извлечения данных, которые требуются для создаваемых ими приложений. Рассматриваемая нами ниже библиотека для поддержки работы с данными на уровне бизнес-сущностей ADO.NET Entity Framework упрощает эти задачи и позволяет разработчикам сфокусироваться на бизнес-логике, а не на нюансах извлечения данных в привязке к конкретной схеме базы данных. Корпоративные системы и бизнес-приложения часто используют данные из различных источников, в которых описаны различные схемы и используются различные соглашения о наименовании объектов. Помимо этого, эти данные используют разные уровни

нормализации, что приводит к тому, что необходимые бизнес-сущности находятся в разных таблицах и записях. В результате, разработчикам приходится писать большие объемы кода для реализации логики приложений, учитывающей комплексные взаимоотношения между данными, находящимися в разных источниках.

Для решения этих задач можно использовать ADO.NET Entity Framework — библиотеку, которая базируется на модели представления бизнес-сущностей **Entity Data Model** (EDM). Эта модель позволяет разработчикам трансформировать реляционные данные и схемы в концептуальные сущности, которые могут использоваться непосредственно в приложениях. Например, данные о заказчиках, используемые в приложении, могут храниться в различных таблицах базы данных. Используя ADO.NET Entity Framework, архитекторы и разработчики могут описать единую сущность «заказчик», которая предоставит уровень абстракции над комплексными связями между таблицами и обеспечит простой механизм получения и обновления данных на уровне приложения. Такой уровень абстракции изолирует логику доступа к данным, превращая ее в набор определенных сущностей, которые могут использоваться на уровне приложения, таким образом, позволяя разработчикам концентрироваться на логике приложения, а не на способах доступа к данным.

Код доступа к данным в существующих приложениях, либо представлен в виде строчных литералов, либо написан на диалекте языка SQL, специфичном для той или иной системы управления базами данных. Из-за этого разработчикам приходится обладать знаниями как языков программирования, так и определенных диалектов языка SQL. Встраивание запросов в виде строчных литералов означает, что такой код не может быть проверен на корректность во время компиляции и разработчикам приходится ждать до тех пор, пока запрос не будет отослан в базу данных и сервер не вернет желаемый результат. Так как при таком подходе схема базы данных и типы недоступны в среде разработчика Visual Studio, технология IntelliSense не применима при работе с объектами базы данных.

Рассматриваемая ниже поддержка возможности задания запросов к базе данных непосредственно в языках программирования C# и Visual Basic .NET — Language Integrated Query (LINQ) — это набор расширений для библиотеки .NET Framework, которые позволяют использовать данные как встроенные типы языка. Технология LINQ позволяет разработчикам писать запросы к данным на том или ином языке программирования, не привязанном к конкретному диалекту SQL, и выполнять проверку типов и синтаксиса во время компиляции всего кода. Так как возможность задания запросов является интегрированной частью языков программирования C# и Visual Basic .NET, разработчикам полностью доступна технология Intelli-Sense, использование которой существенно облегчает и ускоряет написание кода.

Средство разработки — Microsoft Visual Studio — объединяет перечисленные выше технологии работы с данными в продуктивную и простую в использовании среду, рассчитанную как на архитекторов, так и на разработчиков. В Visual Studio поддерживается возможность автоматического создания сущностей для существующих и новых источников данных. Для поддержки нового объектного слоя могут быть сгенерированы сервисные частичные классы (partial classes). После того как создана модель сущностей — Entity Data Model — разработчики могут использовать либо Entity SQL, либо LINQ для написания прикладного кода. И, наконец, командная разработка с использованием Visual Studio Team System существенно улучшает эффективность всего процесса разработки, предоставляя такие возможности, как управление исходным кодом, отслеживание изменений, поддержка развертывания и т. п., которые доступны всей проектной группе — от менеджеров проектов до тестировщиков.

Вторая характеристика платформы для работы с данными компании Microsoft — это *всеобъемность*. Как мы уже отметили выше, данные обычно находятся в различных источниках и, таким образом, разработчикам требуется использовать различные технологии доступа к источникам данных для получения данных, требующихся для обеспечения работы приложения. Помимо этого, современным приложениям и сервисам требуется выполнять сложные задачи по обработке данных, для которых возможности традиционных языков запросов не подходят. Также отметим еще одну проблему — во многих организациях требуется обеспечение поддержки доступа к данным из приложений, которые находятся на различных устройствах и не всегда работают в подключенном режиме.

Три характеристики платформы для работы с данными — широкий набор средств для подключения к источникам данных, богатые технологии программирования и обеспечение доступа к данным в любом месте и на любом устройстве, позволяют решить перечисленные выше проблемы.

Технологии доступа к данным, такие как ODBC и ADO/OLEDB, позволяют разработчикам обращаться к различным базам и источникам данных. ODBC продолжает оставаться наиболее часто используемым способом доступа к источникам данных и поддерживается на уровне Visual Studio для подключения к SQL Server 2008. Другой популярный механизм — ADO/OLEDB позволяет подключаться как к базам данных, так и к другим источникам и также может использоваться для подключения к SQL Server 2008. Microsoft также поддерживает набор технологий для доступа к данным из «управляемых» языков программирования — ADO.NET Entity Framework, провайдеры данных ADO.NET и LINQ. Как мы уже узнали выше, ADO.NET Entity Framework позволяет разработчикам работать с данными на концептуальном уровне, провайдеры данных ADO.NET позволяют обращаться к широкому спектру источников данных, а LINQ обеспечивает запрос и управление данными любого типа непосредственно из языков про-

граммирования. Помимо этого, платформа для работы с данными компании Microsoft поддерживает доступ к различным данным, используя такие технологии, как XML, драйверы JDBC и PHP.

Доступ к данным в любом месте и на любом устройстве обеспечивается поддержкой локальной версии сервера — SQL Server Compact Edition, и использованием механизма синхронизации данных из различных источников — Microsoft Sync Services.

И, наконец, третья характеристика платформы для работы с данными компании Microsoft — *масштабируемость*. Многим организациям требуется обеспечение хранения больших объемов данных в различных форматах, включая реляционные и нереляционные представления данных. Также требуется поддержка хранения таких объектов, как документы, XML-данные, гео-данные и графические изображения. Помимо этого, для поддержки различных типов и большого числа пользователей, обращающихся к данным, необходимо создавать масштабируемые приложения и сервисы, которые смогут обрабатывать растущее число запросов к данным.

SQL Server 2008 доступен в нескольких редакциях, каждое из которых рассчитано на оптимальное обеспечение рабочих нагрузок в определенных сценариях — мобильные решения, клиентские приложения, приложения для небольших групп, отделений и целых компаний.

Как уже говорилось в предыдущих разделах, SQL Server 2008 поддерживает как реляционные, так и нереляционные данные. Используя поддержку .NET Framework CLR в SQL Server, разработчики могут создавать хранимые процедуры и пользовательские типы на «управляемом» коде. Это повышает масштабируемость, так как управляемый код более эффективен и выполняется в непосредственной близости к данным.

Поддержка в SQL Server 2008 различных типов данных, включая GEOMETRY и GEOGRAPHIC для работы с гео-данными и FILESTREAM для обработки BLOB-ов вне хранилища SQL Server, позволяют создавать масштабируемые решения, отвечающие новым сценариям корпоративных приложений. Поддержка соответствующих программных интерфейсов, включая LINQ to XML, делает работу с XML-данными и документами более простой и эффективной.

Выше мы рассмотрели, как SQL Server 2008 и компоненты и технологии, входящие в состав платформы для работы с данными, позволяют создавать масштабируемые решения, отвечающие требованиям к современным приложениям широкого класса. Ниже мы более подробно остановимся на ключевых характеристиках ADO.NET Entity Framework, Language Integrated Query (LINQ) и Microsoft Sync Framework (MSF).

ADO.NET Entity Framework

ADO.NET Entity Framework — это программная библиотека для поддержки работы с данными на уровне бизнес-сущностей. Описанные выше проблемы, связанные с необходимостью в понимании логической схемы данных, необходимостью поддержки работы с различными источниками данных и различными уровнями нормализации в различных источниках данных, в полной мере адресуются ADO.NET Entity Framework за счет предоставления следующих ключевых технологий:

- **Entity Data Model** представляет собой механизм отображения реляционных данных на концептуальные сущности. Использование такой модели позволяет обойтись без изучения структур базы данных и дает разработчикам возможность работать с концептуальными сущностями, которые соответствуют бизнес-объектам и логике приложения.
- Языки запросов **LINQ** и **Entity SQL** позволяют разработчикам писать код доступа к данным непосредственно на .NET-языках, а не на языках и диалектах, специфичных для конкретного источника данных. Это особенно удобно в тех случаях, когда приложению требуется работать с несколькими различными источниками данных.

Модель сущностей — Entity Data Model — позволяет разработчикам отобразить источник данных на сущности, которые используются в приложении. Простым примером таких действий будет отображение сущности «Клиент» на соответствующую таблицу в базе данных, не используя те колонки, которые не требуются бизнес-логике. Во время процесса отображения также можно переименовать колонки для того, чтобы сделать код более читаемым.

Использование Entity Data Model

Адрес для клиента может храниться в отдельной от основных данных о клиенте таблице. Это имеет смысл с точки зрения дизайна базы данных, но для приложения получение обеих типов информации может потребовать создания более сложных запросов на основе команд **JOIN**.

Entity Data Model поддерживает возможность отображения нескольких таблиц на одну сущность — таким образом, используя концептуальную модель, а не логическую модель базы данных, приложение может работать с полноценной сущностью без необходимости в указании, где хранятся отдельные части информации.

Бизнес-сущности часто состоят из групп информации. Например, адрес состоит из номера дома, названия улицы, города и почтового индекса. Entity Data Model позволяет задать комплексные типы, такие как адрес, и затем использовать их при описании бизнес-сущностей.

Отображение логической модели на концептуальную модель описанными выше способами позволяет абстрагировать прикладной код от изменений, которые могут быть внесены в логические структуры на уровне источника данных.

Разработчики часто используют механизмы наследования при работе с однотипными объектами. Например, можно описать объект **Person**, который будет содержать ключевые атрибуты любого человека (фамилия, имя, отчество, пол и т. п.), а на его основе создать объекты **Customer** и **Employee**, которые будут содержать как унаследованные ключевые атрибуты, так и дополнительные атрибуты, уникальные для данного типа объекта. При работе с данными, такой поход реализовать довольно сложно, но модель Entity Data Model поддерживает наследование на уровне концептуальной модели, что существенно упрощает работу с данными.

В Entity Data Model поддерживается три типа наследования:

- *Одна таблица на иерархию*, где одна таблица содержит данные для всех типов и колонка описывает различия между ними;
- *Одна таблица на подкласс*, где одна таблица содержит базовый тип и отдельные таблицы содержат данные для подтипов;
- *Одна таблица* на тип, где одна таблица содержит все данные для подтипа, включая унаследованные данные.

Эти возможности позволяют разработчикам работать с сущностями также, как и с обычными бизнес-объектами.

ADO.NET Entity Framework автоматически генерирует клиентские представления для данных, используя модель Entity Data Model. При необходимости можно настраивать эти представления — для этого существует два способа:

- Получение подмножества данных, используя атрибут **WHERE** в языке запросов Entity SQL:

```
SELECT P
FROM AW.Production.Product AS P
WHERE p.ProductCategory.Name = 'Road Bikes'
```

- Для использования языка запросов, поддерживаемого на уровне хранилища данных, можно использовать атрибут **DefiningQuery**:

```
<EntitySet Name="Product" EntityType="AW.Product">
  <DefiningQuery>
    SELECT p.Name,  p.ProductNumber, p.ReorderPoint
    FROM Product as p WHERE p.DaysToManufacture > 1
  </DefiningQuery>
</EntitySet>
```

Для автоматической генерации модели и файлов отображения (mapping files) можно использовать средства, входящие в состав Visual Studio. Также можно использовать средства Visual Studio или любой другой XML-редактор с поддержкой схем для настройки этих файлов — можно настраивать метаданные или изменять свойства сгенерированных сущностей.

Ниже показаны примеры CSDL и MSL, которые может потребоваться добавить к отображению для использования хранимой процедуры в концептуальной модели.

CSDL

```
<FunctionImport Name="GetOrderDetails"
    EntitySet="SalesOrderDetail"
    ReturnType="Collection(AdventureWorksModel.SalesOrderDetail)">
   <Parameter Name="SalesOrderHeaderId" Type="Int32" Mode="in">
   </Parameter>
</FunctionImport>
```

MSL

```
<FunctionImportMapping FunctionImportName="GetOrderDetails"
           FunctionName="AdventureWorksModel.Store.GetOrderDetails"/>
```

Помимо этого, можно расширять и автоматически генерируемый .NET-код. Соответствующие средства Visual Studio создают частичные классы для объектов в модели и поддерживается возможность включения в них любой требующейся для функционирования приложения бизнес-логики. Частичные классы также используются для того, чтобы отделить сгенерированный код от кода разработчика и при необходимости повторной генерации кода код разработчика не будет потерян.

Помимо адаптивности и расширяемости — двух ключевых характеристик ADO.NET Entity Framework, которые мы кратко рассмотрели выше, ADO.NET Entity Framework — это развивающаяся библиотека. В состав Visual Studio 2008 входит набор утилит, которые позволяют разработчикам работать с моделями сущностей и соответствующими отображениями на источники данных. К таким утилитам относятся:

- Дизайнер сущностей **Entity Designer**, который используется для генерации моделей, редактирования свойств и проверки моделей и отображений;
- Утилита **Entity Mapping** для задания сущностей и ассоциаций;
- Утилита **Entity Model Browser** для визуализации модели.

ADO.NET Entity Framework включает новый провайдер для доступа к данным на уровне ADO.NET, который используется для работы с концептуальными моделями, задаваемыми на уровне библиотеки. Этот провайдер, известный под названием **EntityClient**, позволяет разработчикам использовать знакомые концепции провайдеров данных, при этом учитывая изменения, привносимые Entity Data Model.

Использование двух новых языков доступа к данным — LINQ и Entity SQL, позволяет задавать запросы и получать результаты в виде типизованных CLR-объектов. Использование данных в виде типизованных CLR-объектов обеспечивает лучшую функциональность на уровне средств разработки и дизайна и обеспечивает полную поддержку технологии IntelliSense. Помимо этого, типизация позволяет обнаружить ошибки во время компиляции, а не во время выполнения, что сокращает время, требуемое на отладку и, таким образом, упрощает разработку приложений.

Использование ADO.NET Entity Framework также помогает снизить время и затраты на сопровождение:

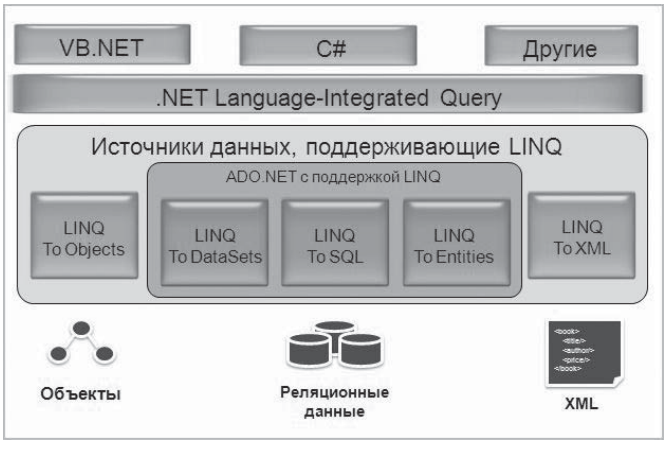
- Изменения в базе данных не всегда приводят к необходимости внесения изменений в код приложения — в ряде случаев достаточно внести соответствующие изменения в модель и файлы отображения логической схемы на сущности.
- Так как код приложения привязан к концептуальной модели, а не к конкретному типу базы данных, переход на новую версию SQL Server может быть более простым.

Выше мы рассмотрели основные возможности ADO.NET Entity Framework, которые могут существенно облегчить и упростить создание широкого класса приложений, работающих с данными, хранящимися в базах данных. Более подробную информацию об ADO.NET Entity Framework можно получить в разделе, посвященном этой технологии на сайте MSDN .Далее мы обратимся к основным характеристикам встроенного языка запросов LINQ.

Language Integrated Query (LINQ)

LINQ — это поддержка возможности задания запросов к базе данных непосредственно в языках программирования C# и Visual Basic .NET. Если отложить в сторону теоретические рассуждения о важности и полезности

наличия унифицированного механизма доступа к объектам, реляционным данным и XML-документам, и кратко описать данную технологию, то LINQ позволяет решить две задачи — во-первых, унифицировать механизм запросов к различным объектам, пусть то будут объекты типа массивов, коллекций и т. п., находящиеся в памяти, базам данных и XML-документам и, во-вторых, интегрировать средства написания таких запросов непосредственно в язык программирования. LINQ представляет собой набор расширений для языков C# 3.0 и Visual Basic .NET 9.0, а также унифицированную программную модель на уровне библиотеки классов .NET Framework. Ниже мы посмотрим, как LINQ поддерживается в языке C# 3.0 — изучение аналогичных возможностей языка Visual Basic .NET 9.0 мы оставим читателям в виде самостоятельного упражнения.



Технология LINQ

Мы сосредоточимся на рассмотрении использования LINQ для работы с источниками реляционных данных (LINQ to SQL), но сначала кратко опишем возможности LINQ для работы с объектами и XML-документами.

Наше знакомство с технологией LINQ мы начнем с рассмотрения механизмов доступа к объектам (т. н. OLINQ или Object LINQ). Любая коллекция, поддерживающая интерфейс System.Collections.Generic.IEnumerable или более общий интерфейс IEnumerable<T>, может рассматриваться в виде последовательности данных и, таким образом, может быть обработана, используя стандартные операторы запросов LINQ. Эти стандартные операторы позволяют разработчикам создавать запросы, включая возможность создания проекций в виде новых типов. Эта функциональность используется совместно с рассмотренной в предыдущем разделе возможностью автоматического определения типов переменных по их инициализации.

Механизмы доступа к XML (т. н. XLINQ или XML LINQ) позволяют работать с кэшем XML, находящимся в памяти, а также предоставляют про-

стые способы создания XML-документов и их фрагментов. В следующем примере мы посмотрим, как считывать XML-документы в объект XDocument, как выполнять запросы к элементам этого объекта и как создавать документы и их элементы на лету.

Ниже мы обсудим, как использовать механизм LINQ to SQL, который является частью технологии LINQ. Механизм LINQ to SQL позволяет выполнять запросы и манипулировать объектами, ассоциированными с таблицами баз данных. Использование этого механизма позволяет избежать традиционных разногласий между таблицами, хранящимися в базами данных и объектами, представляющими бизнес-логику.

Для создания объектной модели для базы данных, классы должны быть приведены в соответствие с сущностями, хранящимися в базе данных. Можно выделить три способа реализации такого приведения – можно задавать атрибуты для существующих объектов, можно использовать специальное средство, позволяющее автоматически сгенерировать объекты и использовать утилиту командной строки SQLMetal. В наших примерах мы будем использовать первые два из описанных способов.

Начнем с того, что добавим к коду нашего примера ссылки на два пространства имен, которые позволят нам использовать механизм LINQ to SQL:

```
using System.Data.Linq;
using System.Data.Linq.Mapping;
```

Теперь добавим к классу Customer ряд атрибутов для того, чтобы привести его в соответствие с таблицей базы данных Customers, которая содержит поля CustomerID и City. Ниже показано, как привести наш класс в соответствие с таблицей Customers в базе данных Northwind:

```
[Table(Name = "Customers")]
public class Customer
{
 [Column]
 public string CustomerID { get; set; }
 [Column]
 public string City { get; set; }

 public override string ToString()
 {
  return CustomerID + "\t" + City;
 }
}
```

Теперь обратимся к методу ObjectQuery() и преобразуем метод поиска клиентов, находящихся в Лондоне — выше мы уже видели, как это сделать для данных, расположенных в памяти и в XML-файле. Обратим внимание

на то, что нам потребуется внести минимальные изменения для того, чтобы наш запрос мог быть обращен к данным, находящимся в базе данных. После того как мы создали соединение с базой данных, мы можем получить необходимые нам записи из таблицы Customers, выбрать те записи, которые содержат клиентов, находящихся в Лондоне, и вернуть их в виде IEnumerable<Cumstomer>. Вот код модифицированного метода ObjectQuery():

```
static void ObjectQuery()
{
 DataContext db = new DataContext
   (@"Data Source=.\sqlexpress;Initial Catalog=Northwind");
   var results = from c in db.GetTable<Customer>()
        where c.City == "London"
        select c;
   foreach (var c in results)
    Console.WriteLine("{0}\t{1}", c.CustomerID, c.City);
}
```

Запустим наше приложение и убедимся в том, что оно работает, как ожидалось. В качестве дополнительной опции мы можем добавить код, показывающий SQL-запрос, выполняемый для получения данных. Вставьте следующую строку:

```
db.Log = Console.Out;
```

сразу после объявления переменной DataContext.

Теперь давайте посмотрим, как выполнить те же действия, но используя автоматически созданный объект, отраженный на таблицу базы данных. Прежде всего, удалим из нашего примера описание класса Cumstomer. Затем выберем команду Add | New Item для добавления к проекту нового элемента и в списке Templates выберем шаблон LINQ To SQL File. В поле Name укажем имя Northwind и щелкнем кнопку Ok. Выберем команду View | Server Explorer (или нажмем последовательность клавиш Crtl+W, L), затем щелкнем кнопку Connect to database. В диалоговой панели Add Connection в поле Server name укажем имя сервера баз данных (например, .\sqlexpress), выберем базу данных Northwind в списке Select or enter a database name и щелкнем кнопку Ok.

Следующий шаг — это создание объекта представления. Откроем дерево под названием Data Conenctions, затем — папку Northwind, затем — папку Tables и откроем файл Northwind.dbml. Из папки с таблицами перетащим на панель методов таблицы Customers, Products, Employees и Orders. Затем из папки с хранимыми процедурами перетащим на панель методов хранимую процедуру Top Most Expensive Products. Нажмем комбинацию клавиш Ctrl+Shift+B для сборки приложения. Посмотрим на автоматичес-

ки сгенерированный класс и обратим внимание на использование атрибутов. Отметим, что для баз данных с большим числом таблиц и хранимых процедур утилита SQLMetal предоставляет большее число возможностей для генерации объекта представления.

Вернемся к нашему методу ObjectQuery(). Каждая таблица представляет собой соответствующее свойство переменной db. В нашем примере запрос к базе данных будет практически аналогичным запросу, написанному в предыдущих примерах. Добавим в метод ObjectQuery()следующий код для получения списка клиентов, расположенных в Лондоне:

```
static void ObjectQuery()
{
  var db = new NorthwindDataContext();
  db.Log = Console.Out;
  var results = from c in db.Customers
            where c.City == “London”
                 select c;
    foreach (var c in results)
     Console.WriteLine(“{0}\t{1}”, c.CustomerID, c.City);
}
```

В приведенном выше примере мы создали объект NorthwindDataContext, который представляет собой строго типизированное соединение с базой данных. Отметим, что мы в явном виде не используем строку соединения с базой данных.

Запустим приложение на выполнение и убедимся в том, что оно работает, как и предполагалось. Используя дизайнер, можно создавать отображения и на другие таблицы. Класс Customer использует отображение «один ко многим» на таблицу Orders. В следующем запросе показано, как извлечь данные из нескольких таблиц:

```
static void ObjectQuery()
{
  var db = new NorthwindDataContext();
  db.Log = Console.Out;
  var results = from c in db.Customers
                from o in c.Orders
                 where c.City == “London”
                select new { c.ContactName, o.OrderID };
  foreach (var c in results)
  Console.WriteLine(“{0}\t{1}”, c.ContactName, c.OrderID);
}
```

В приведенном выше примере показана конструкция select, создающая новый объект анонимного типа (новая возможность в C# 3.0). Созданный

тип содержит два элемента данных — две строки с названиями, соответствующими оригинальным данным (в нашем случае — это ContactName и OrderID). Использование анонимных типов чрезвычайно полезно при работе с запросами и существенно сокращает время на создание классов, содержащих результаты выполнения запросов.

В следующем примере мы рассмотрим, как, используя технологию LINQ, мы можем не только извлекать данные, но и выполнять операции создания, обновления и удаления, т. е. выполнять все четыре типа операций, которые обычно называют CRUD-операции (Create, Read, Update, Delete).

Добавим новый метод для модификации данных в базе данных — modifyData() и соответствующий вызов в методе Main():

```
static void Main(string[] args)
{
  modifyData();
}

static void modifyData()
{
  var db = new NorthwindDataContext();
  var newCustomer = new Customer
  {
   CompanyName = “AdventureWorks Cafe”,
   CustomerID = “ADVCA”
  };

  db.Customers.Add(newCustomer);

  Console.WriteLine(“Number Created:{0}”,
  db.Customers.Where( c => c.CustomerID == “ADVCA” ).Count());
}
```

Выполним наше приложение и отметим, что на экране показан 0, а не 1, и новая запись не отображена в результатах. Это связано с тем, что мы еще не вызвали метод submitChanges(). Теперь изменим данные в базе данных — модифицируем контактную информацию для первого клиента:

```
static void modifyData()
{
 var db = new NorthwindDataContext();
 var newCustomer = new Customer
 {
  CompanyName = “AdventureWorks Cafe”,
  CustomerID = “ADVCA”
 };
```

```
 db.Customers.Add(newCustomer);

 Console.WriteLine("Number Created:{0}",
 db.Customers.Where( c => c.CustomerID == "ADVCA" ).Count());

 var existingCustomer = db.Customers.First();

 existingCustomer.ContactName = "New Contact";
 Console.WriteLine("Number Updated:{0}",
 db.Customers.Where( c => c.ContactName == "New Contact" ).Count());
}
```

Как и в предыдущем примере, на экране будет показан 0, так как мы все еще не вызвали метод submitChanges(). Для того чтобы локальные изменения были занесены в базу данных, добавим к нашему методу modify Data()следующий код:

```
 db.SubmitChanges();

 Console.WriteLine("Number Created:{0}",
 db.Customers.Where( c => c.CustomerID == "ADVCA" ).Count());

 Console.WriteLine("Number Updated:{0}",
 db.Customers.Where( c => c.ContactName == "New Contact" ).Count());
```

Выполним наше приложение и убедимся в том, что оно работает, как и ожидалось. Отметим, что после того как новый клиент занесен в базу, он не может быть занесен еще раз из-за ограничений на первичные ключи, поэтому данный пример можно запустить только один раз.

Теперь посмотрим, как с помощью технологии LINQ мы можем вызывать хранимые процедуры — помните, что в дизайнере мы добавили одну хранимую процедуру? Создадим новый метод, который будет выводить результаты выполнения хранимой процедуры Top Most Expensive Products:

```
static void InvokeSproc()
{
 var db = new NorthwindDataContext();
 foreach (var r in db.Ten_Most_Expensive_Products())
 Console.WriteLine(r.TenMostExpensiveProducts + "\t" + r.UnitPrice);
}
```

Изменим код метода Main() на следующий:

```
static void Main(string[] args)
{
    InvokeSproc();
}
```

Выполним наше приложение и изучим результаты. Отметим, что когда по каким-то причинам база данных не может быть использована через динамические запросы на языке SQL, мы можем воспользоваться языком C# 3.0 и технологией LINQ для вызова хранимых процедур, предоставляющих доступ к данным.

Запросы, которые мы выполняли выше, в основном выполняли операции фильтрации данных, но возможности LINQ этим не ограничиваются. Так, например, для сортировки клиентов, находящихся в Лондоне, мы можем использовать конструкцию orderby:

```
static void ObjectQuery()
{
 var db = new NorthwindDataContext();
 db.Log = Console.Out;
 var results = from c in db.Customers
               where c.City == "London"
               orderby c.ContactName descending
               select new { c.ContactName, c.CustomerID };
 foreach (var c in results)
  Console.WriteLine("{0}\t{1}", c.CustomerID, c.ContactName);
}
```

Для того чтобы найти число клиентов, находящихся в каждом городе, мы можем использовать конструкцию group by:

```
static void ObjectQuery()
{
 var db = new NorthwindDataContext();
 db.Log = Console.Out;
 var results = from c in db.Customers
               group c by c.City into g
               orderby g.Count() ascending
               select new { City = g.Key, Count = g.Count() };
 foreach (var c in results)
 Console.WriteLine("{0}\t{1}", c.City, c.Count);
}
```

Конструкция group by создает переменную типа IGrouping<string, Customer>, где строка содержит название города.

Часто при написании запросов возникает необходимость в поиске по двум таблицам. Обычно для этих целей используется операция join, поддерживаемая в LINQ и C# 3.0. В методе ObjectQuery() заменим существующий запрос на приведенный ниже. Вспомним, что мы отображали все заказы для каждого клиента, находящегося в Лондоне. Теперь же мы отобразим число заказов, размещенных каждым клиентом.

```
static void ObjectQuery()
{
 var db = new NorthwindDataContext();
 db.Log = Console.Out;
 var results = from c in db.Customers
               join e in db.Employees on c.City equals e.City
               group e by e.City into g
               select new { City = g.Key, Count = g.Count() };
 foreach (var c in results)
 Console.WriteLine("{0}\t{1}", c.City, c.Count);
}
```

В этом разделе мы рассмотрели основные возможности технологии LINQ, которая может использоваться для работы с данными, хранимыми в структурах и объектах, в базах данных и XML-документах. Более подробно о технологии LINQ и ее возможностях см. раздел, посвященный этой технологии на сайте MSDN.

Microsoft Sync Framework (MSF)

Microsoft Sync Framework (MSF) — это механизм синхронизации данных, позволяющий разработчикам добавить поддержку синхронизации на уровне приложений, сервисов и устройств. MSF позволяет решить фундаментальную проблему синхронизации любых типов данных, используя любой сетевой протокол. На сегодняшний день, Microsoft Sync Framework — это единственное доступное на рынке решение, предлагающее открытую архитектуру, широкий набор возможностей и поддержку синхронизации «точка-точка» (peer-to-peer synchronization) между устройствами и клиентскими приложениями, между устройствами и серверами и т. п.

Давайте рассмотрим ряд наиболее востребованных сценариев, связанных с синхронизацией, и проблемы, которые возникают при их реализации. К таким сценариям относятся:

- Кэширование в стиле Microsoft Outlook для поддержки как онлайновых, так и отсоединенных операций;
- Сценарии поддержки различных устройств и таких данных, как персональные данные и настройки, аудио, видео, файлы/папки, на устройствах типа персональных компьютеров, на уровне сервисов и на мобильных устройствах, включая синхронизацию «точка-точка» или через «посредников» в виде компьютеров или сервисов;
- Поддержка отсоединенных режимов работы для веб-сервисов и веб-приложений;
- Кэширование справочников для бизнес-приложений на клиентских компьютерах с возможностью работы таких приложений в отсоединен-

ном режиме и поддержкой синхронизации изменений внесенных на клиенте с централизованным хранилищем данных.

Сценарии синхронизации

К проблемам реализации перечисленных выше сценариев синхронизации можно отнести:

- Необходимость в поддержке большого числа устройств и сервисов доступа к данным;
- Необходимость в поддержке большого числа форматов и протоколов;
- Комплексные сценарии обмена данными, особенно при их обновлении;
- Повышение спроса на сценарии поддержки мобильных устройств и отсоединенных режимов работы.

Задачи синхронизации

Помимо этого, ряд задач, связанных с синхронизацией, требует существенных ресурсов для реализации. Особенно это относится к :

- Определению данных, которые необходимо синхронизировать;
- Разрешению конфликтов, возникающих при синхронизации;
- Адекватной реакции на возникающие ошибки;
- Работе в сетевом окружении;
- Синхронизации из нескольких источников;
- Добавлению новых источников в систему синхронизации данных.

Microsoft Sync Framework (MSF) позволяет решить большинство из перечисленных выше проблем. При создании MSF разработчики преследовали следующие цели:

- Создать среду выполнения общего назначения, которая была бы надежной и оптимизированной и позволяла бы создавать синхронизационные решения с поддержкой кэширования, отсоединенного режима работы, совместного доступа к данным и обмена информацией;
- Облегчить обмен данными даже при использовании различных протоколов и типов хранилищ за счет стандартизации синхронизационных метаданных;
- Упростить разработку синхронизационных решений за счет использования специфичных компонентов для реализации типовых сценариев — таких как синхронизация реляционных баз данных, файловых систем, списков, устройств, музыки, видео и т. п.

Microsoft Sync Framework поддерживает различные топологии синхронизации, включая peer-to-peer, hub/spoke и full-mesh и использует существующие протоколы и механизмы передачи данных для обеспечения простых, но эффективных механизмов синхронизации. В отличие от других синхронизационных решений, которые фокусируются на синхронизации содержимого, MSF поддерживает синхронизацию на уровне метаданных. *Метаданные* — это подмножество информации, описывающей содержимое, документ, или неструктурированные данные.

Синхронизация на уровне метаданных представляет собой более надежный способ синхронизации между различными хранилищами данных и позволяет MSF решить ключевые задачи синхронизации и обеспечить универсальный обмен данными. Используя высокоэффективное, компактное представление метаданных для поддержки версионности и быстрого обнаружения изменений, MSF позволяет практически любому устройству, сервису, приложению или платформе выполнять полную, многоцелевую синхронизацию. В состав MSF входит ряд уникальных алгоритмов, позволяющих выполнять синхронизацию в различных режимах, включая параллельные обновления, обнаружение конфликтов в любой ветви и распространять необходимые обновления по всей топологии. Использование метаданных позволяет, в частности, обеспечивать поддержку синхронизации на основе фильтров, т. е. можно задавать такие критерии синхронизации, как

«все песни с рейтингом 4 звезды», «почта, полученная за последние 2 дня» или синхронизировать данные на основе значений подмножества полей.

С точки зрения архитектуры, Microsoft Sync Framework состоит из трех основных компонентов — *ядра синхронизации*, используемого для управления операциями в режиме выполнения, *компонентов*, обеспечивающих механизмы синхронизации, и *инфраструктуры*, позволяющей быстро создавать типовые синхронизационные решения на базе SQL Server, SQL Server Compact Edition, файловых систем FAT и NTFS и т. п. Рассмотрим ключевые компоненты Microsoft Sync Framework более подробно. Начнем с ядра синхронизации.

Ядро синхронизации содержит сервисы метаданных, используемые всеми клиентами MSF. К основным функциональным возможностям ядра относятся:

- Средства представления и управления метаданными, включая обнаружение конфликтов, встроенные или расширяемые пользователями механизмы разрешения конфликтов, средства обнаружения изменений и т. п.;
- Средства управления конфликтами при синхронизации и разрешения конфликтов;
- Средства отслеживания изменений — как на уровне элементов и свойств, так и на уровне групп;
- Обработка фильтров и синхронизация на уровне фильтров;
- Восстановление после сбоев, включая такие сценарии, как прерывание операций, сбой сети и т. п.;
- Управление сессиями синхронизации, включая отмену операций, отображение прогресса и т. п.

В состав Microsoft Sync Framework входит агент синхронизации — Synchronization Agent, который может использоваться для синхронизации по запросу в режиме «точка-точка». Точки представляют собой абстракции, которые поддерживаются на уровне провайдеров (Sync Provider), которые отвечают за функциональность конкретной точки, хранилища или протокола. Приложения используют провайдеров для обеспечения синхронизации, а разработчики могут создавать собственных провайдеров для практически любых типов данных.

Ключевым компонентом Microsoft Sync Framework также является поддержка Simple Sharing Extensions (SSE). В MSF входит поддержка точек, которые могут взаимодействовать на основе расширений SSE для поддержки протоколов RSS и ATOM. Более того, Microsoft Sync Framework обеспечивает сервисы для потребления и генерации потоковых данных, включая обнаружение конфликтов и связанные с этим сервисы. Поддержка RSS/ATOM включена в состав MSF. Но при необходимости разработчики могут добавить поддержку и других потоковых протоколов.

Второй компонент Microsoft Sync Framework — это набор компонентов или *провайдеров*, обеспечивающих механизмы синхронизации. Эти провайдеры непосредственно связаны с *компонентами*, обеспечивающими доступ к конкретным типам хранилищ. К таким провайдерам относятся:

- **Провайдеры для реляционных данных.** Synchronization Services for ADO.NET обеспечивают возможность синхронизировать реляционные данные из различных источников. Поддерживаются двухзвенные, n-звенные и основанные на сервисах архитектуры. Программные интерфейсы Synchronization Services API, модель которых схожа с программными интерфейсами доступа к данным ADO.NET, обеспечивают необходимые интерфейсы для синхронизации данных, особенно в тех сценариях, когда мобильные пользователи работают с локальными копиями данных в отсоединенных режимах.
- **SQL Server Compact Metadata Store** — компонент, который может использоваться для хранения синхронизационных метаданных, таких как версии, привязки и информации об обнаруженных изменениях. Данный компонент существенно облегчает создание собственных провайдеров, особенно в тех случаях, когда требуется обеспечение хранения метаданных.
- **File and Folder Sync Provider** — провайдер для работы с любой файловой системой, совместимой с файловой системой Win32 — FAT, NTFS, переносные устройства и т. п. Данный провайдер обеспечивает такие функции, как обнаружение изменений в FAT-томах (включая перемещения и переименования объектов), разрешение коллизий на уровне имен, разрешение конфликтов при обновлениях/удалениях (включая иерархические операции) и возможность предварительного просмотра результатов синхронизации.
- **Microsoft Codename «Astoria» Offline Provider** — обеспечивает синхронизацию на основе REST-интерфейсов для онлайновых сервисов доступа к данным в рамках проекта «Astoria» (ADO.NET Data Services).

После того как мы рассмотрели основные компоненты Microsoft Sync Framework, давайте посмотрим на то, как MSF позволяет создавать гибкие синхронизационные решения.

Как мы уже отметили выше, MSF можно использовать для реализации синхронизационных сценариев для практически любых приложений, сервисов или устройств — от корпоративных приложений, использующих базы данных, до USB-устройств.

Основанная на метаданных синхронизация позволяет синхронизировать данные любого типа — от персональной информации до цифровых мультимедийных данных.

Использование существующих архитектур и протоколов для синхронизации данных обеспечивается на уровне MSF за счет «прозрачных» механизмов управления обменом данными из различных источников.

Синхронизационные решения для любой платформы

Помимо полноценных провайдеров (т. н. «полных участников»), которые поддерживают сценарии синхронизации peer-to-peer и full mesh, MSF обеспечивает поддержку участия ряда других элементов в обеспечении синхронизации. В частности, провайдеры MSF Sync Providers обеспечивают реализацию следующих сценариев:

- **Частичное участие** — обеспечивается провайдерами, которые просто сохраняют информацию, но не поддерживают синхронизационные метаданные — простые USB-устройства, телефоны старых моделей, мультимедийные устройства.
- **Простое участие** — обеспечивается провайдерами для точек, которые не поддерживают возможность отслеживать изменения и не могут хранить метаданные.
- **Привязки** — обеспечивается провайдерами для точек, поддерживающих простые механизмы обнаружения изменений на основе временных отметок, отсчета времени и т. п.

Microsoft Sync Framework распространяется бесплатно для использования на платформах Windows и Windows Mobile. В ряде случаев, при включении в экосистему синхронизации платформ, сервисов или устройств, непосредственно не производимых Microsoft, могут потребоваться дополнительные лицензии.

Заключение

В данном разделе мы рассмотрели ряд технологий, непосредственно связанных с Microsoft SQL Server 2008, и позволяющих более эффективно создавать различные классы приложений, использующие данный сервер — ADO.NET Entity Framework, Language Integrated Query (LINQ) и Microsoft Sync Framework (MSF).

Следующий раздел будет посвящен новым возможностям Microsoft SQL Server 2008 в области аналитики, хранилищ данных и новым характеристикам таких сервисов, как Integration Services, Reporting Services и Analysis Services.

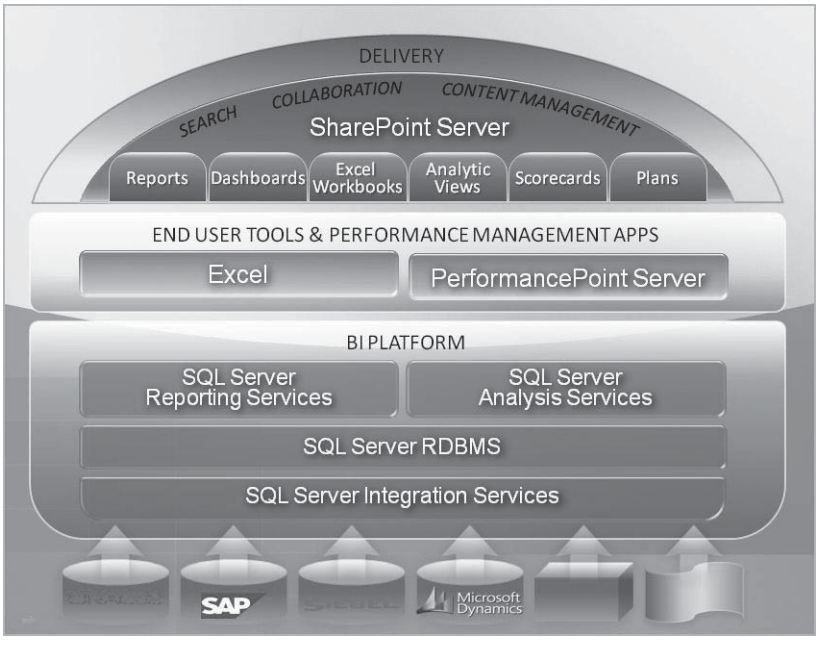
Глава 5

Интуитивное представление

Введение

В данном разделе мы рассмотрим ряд технологий и сервисов, реализованных в Microsoft SQL Server 2008 и предназначенных для интеграции данных из различных источников, анализа данных, создания отчетов и хранилищ данных. Такие технологии объединяются единым названием Business Intelligence и Data Warehouse и поддерживаются на уровне следующих сервисов, включенных в состав продукта — Integration Services, Analysis Services и Reporting Services.

На следующем рисунке показан весь набор решений компании Microsoft, предназначенных для анализа данных, создания OLAP- и BI-решений, включая компоненты, относящиеся непосредственно к SQL Server — они показаны в нижней части рисунка и объединены термином BI Platform — и компоненты, предназначенные для конечных пользователей, которые включают клиентские и серверные продукты семейства Microsoft Office.

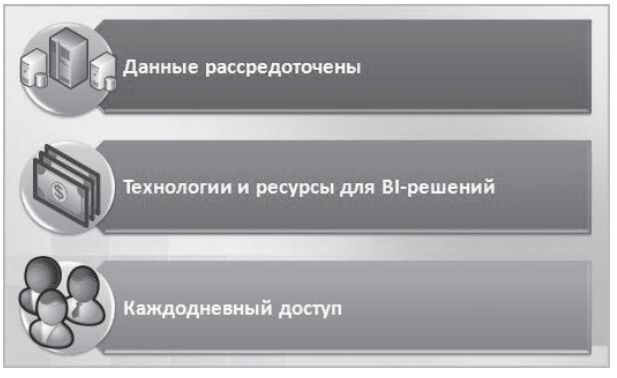


Набор решений компании Microsoft, предназначенных для анализа данных

Можно выделить три основные проблемы, с которыми сталкиваются компании, пытающиеся реализовать решения класса Business Intelligence:

- Бизнес-данные рассредоточены по всему предприятию;
- Создание и управление решениями класса Business Intelligence требует использования различных технологий и привлечения большого числа ресурсов;

- Доступ к бизнес-данным должен обеспечивать выполнение каждодневных задач, и поддерживать не только специализированные отчеты и анализы.

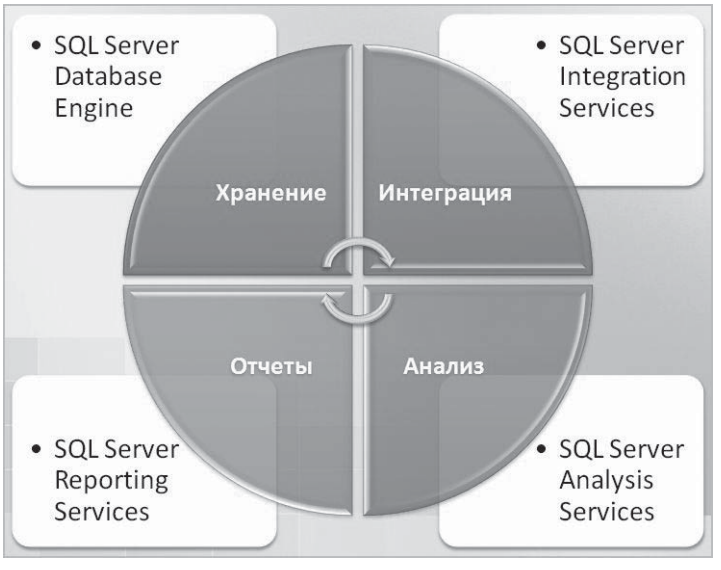


Основные проблемы при реализации BI-решений

Входящие в состав SQL Server 2008 средства позволяют максимально упростить создание решений класса Business Intelligence и решить перечисленные выше задачи. Выделим основные характеристики SQL Server 2008, позволяющие решать задачи класса Business Intelligence, а затем рассмотрим каждую характеристику более подробно.

- Унифицированное хранилище данных
  - Возможность консолидации корпоративных данных для анализа и создания отчетов;
  - Высокая производительность хранилищ данных;
  - Высокая производительность средств анализа данных;
  - Масштабируемость решений для генерации отчетов.
- Возможность создания и управления BI-решениями любой сложности
  - Повышение производительности разработчиков;
  - Снижение затрат на управление и сопровождение BI-решений.
- Повышенная масштабируемость и доступность
  - Доступ к бизнес-данным и аналитике, используя знакомые средства;
  - Гибкие решения с использованием отчетов.

Основные сервисы, включенные в состав SQL Server 2008 и обеспечивающие перечисленные выше возможности — это SQL Server 2008 Database Engine, SQL Server Integration Services, SQL Server Analysis Services и SQL Server Reporting Services.



Компоненты SQL Server 2008 для Business Intelligence

Унифицированное хранилище данных

В большинстве организаций используются многочисленные бизнес-системы, каждая из которых использует собственное хранилище данных. Несмотря на то что имеется возможность генерации отчетов на уровне отдельных приложений и выполнения анализа содержащихся в этих приложениях данных, вы сможете получить представление о всем бизнесе только после того, как выполните консолидацию данных из различных источников для создания централизованного хранилища бизнес-данных для их анализа и генерации отчетов.

В SQL Server 2008 поддерживается два типовых подхода к унификации бизнес-данных для анализа и генерации отчетов:

- **Хранилище данных** (Data warehouse) — специализированное хранилище для данных, которое заполняется и синхронизируется с бизнес-данными, хранящимися в различных источниках;
- **Абстракция источников данных** (Data source abstraction) — Analysis Services в SQL Server 2008 позволяет создавать представления для источников данных (data source views) для реализации уровня абстракции над одним или более реальным источником данных. После создания уровня абстракции, он может использоваться как единый источник данных для Analysis Services, Integration Services и Reporting Services.

В состав SQL Server 2008 входит расширенная поддержка хранения XML-данных, а также новый тип данных FILESTREAM, использование которого позволяет хранить бинарные данные большого объема в файловой сис-

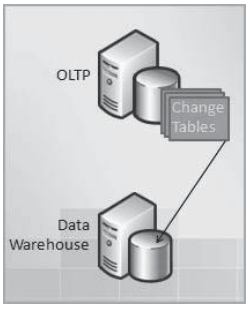
теме — в этом случае такие данные остаются частью базы данных и работа с ними осуществляется в рамках транзакций на уровне базы данных. Помимо этого в SQL Server 2008 сняты ограничения на размер пользовательских типов данных (user-defined data types).

Упомянутые выше представления для источников данных позволяют реализовать интеграцию и доступ к данным, расположенным в гетерогенных источниках — таких как SQL Server, Oracle, DB2 и Teradata. Помимо этого, представления для источников данных позволяют обеспечить масштабируемое OLAP-хранилище с поддержкой различных источников данных на уровне сервисов интеграции — SQL Server Integration Services.

Новая версия Microsoft SQL Server — SQL Server 2008 представляет собой платформу для создания масштабируемых хранилищ данных, которые могут использоваться компаниями для интеграции данных в хранилище и, таким образом, обеспечить лучшую масштабируемость и поддержку растущих объемов данных и увеличивающегося числа пользователей.

Хранилища данных обычно заполняются и обновляются данными, получаемыми из исходных систем с использованием процесса, который состоит из трех шагов — извлечение, преобразование и загрузка (Extract, Transform, Load — ETL). После начальной загрузки данных в хранилище, выполняются периодические обновления и изменения данных, получаемых их исходных систем — таким образом, обеспечивается актуальность данных, находящихся в хранилище.

Интеграционные сервисы SQL Server Integration Services представляют собой платформу, средства которой могут использоваться для извлечения данных из различных исходных систем, выполнения необходимых преобразований в соответствии со структурой и форматами, принятыми в хранилище, и загрузки преобразованных данных в хранилище данных. Выполнение операций lookup для нахождения соответствий между исходными системами и записями в хранилище данных является наиболее частой задачей, выполняемой в процессе ETL. В SQL Server 2008 производительность операций lookup существенно улучшена — теперь такие операции масштабируются для поддержки таблиц больших объемов.



Change Data Capture (CDC)

Для упрощения отслеживания изменений данных и обеспечения целостности данных в хранилище, в SQL Server 2008 реализована новая функция — Change Data Capture (CDC) — с ее помощью протоколируются изменения в таблицах, что существенно облегчает нахождение записей, содержимое которых изменилось, а также получение информации о деталях, связанных с модификациями и причинах изменений.

Поддержка сжатия данных, реализованная в SQL Server 2008, позволяет более эффективно хранить данные и снизить затраты на собственно хранилище. Помимо этого, сжатие данных существенно улучшает производительность в сценариях, когда выполняется большое число операций ввода/вывода. В SQL Server 2008 также поддерживается сжатие резервных копий, а использование типа данных VARDECIMAL и новых разреженных колонок (sparse columns) помогают снизить требования к ресурсам, используемым для хранения данных, особенно в тех случаях, когда требуется хранение больших таблиц, содержащих большое число численных или NULL-значений, что часто встречается в хранилищах данных.

В дополнение к эффективным алгоритмам хранения данных, в SQL Server 2008 реализована поддержка динамического распределения AWE-памяти. Поддерживается до 64 Гбайт памяти при работе на платформе Windows Server 2003 Datacenter Edition и до 2 Тбайт при использовании 64-битной версии сервера — это позволяет создавать хранилища данных очень больших объемов.

Новый компонент SQL Server 2008 — Resource Governor — позволяет администраторам задавать ограничения по использованию ресурсов и приоритеты для различных типов загрузок (workload), обеспечивая, таким образом, параллельное выполнение нескольких типов загрузок и оптимальное распределение ресурсов с возможностью проактивного устранения возникающих проблем.

В состав SQL Server 2008 входит высокопроизводительное ядро реляционной базы данных, на основе которого можно создавать высокоэффективные решения с использованием хранилищ данных. Такие новинки, как оптимизация запросов для схемы «звезда» и средства, позволяющие настраивать индексы и структуры данных, делают SQL Server наиболее подходящей платформой для создания хранилищ данных с высоким числом запросов. Новая конструкция языка Transact-SQL — **MERGE** — позволяет разработчикам более эффективно обрабатывать наиболее частые сценарии работы с хранилищами данных — выполнять проверку наличия записи перед выполнением операций вставки или обновления данных. Более подробно о конструкции **MERGE** см. раздел «Новая функциональность в T-SQL».

Помимо этого, расширения конструкции **GROUP BY** позволяют задавать несколько групп в рамках одного запроса. Результат выполнения такого запроса представляется в виде единого набора данных, что является эквивалентом выполнения операции **UNION ALL** для записей в различ-

ных группах и позволяет более быстро выполнять агрегации, требуемые для получения данных и генерации отчетов.

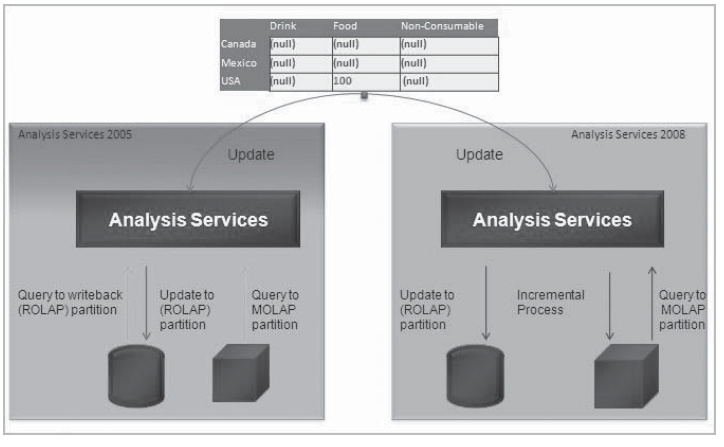
Поддержка деления таблиц (partitioned tables) в SQL Server 2008 позволяет упростить оптимизацию, повысить производительность и управляемость в сценариях, когда используются таблицы большого размера, а новая функциональность — параллелизм при использовании деления таблиц — позволяет существенно оптимизировать хранилище данных.

Сервисы анализа — Analysis Services, входящие в состав SQL Server 2008, позволяют выполнять комплексный анализ данных, включая данные с многочисленными размерностями и агрегациями. Производительность сервисов анализа достигается за счет включения в состав SQL Server 2008 следующих функций:

- **Гибкая модель кэширования.** Используя Analysis Services, вы контролируете кэширование данных и агрегатов для того, чтобы максимально оптимизировать выполнение запросов, но при этом сохраняя приемлемый уровень задержек между кэшем и хранилищем данных.
- **Декларативное взаимоотношение между атрибутами.** В размерности Analysis Services имеется возможность непосредственного задания взаимоотношений между атрибутами иерархии. Это позволяет Analysis Services заранее сгенерировать агрегаты для обработки куба или размерности, что улучшает производительность выполнения запросов.
- **Блочные вычисления.** Блочные вычисления позволяют отказаться от лишних вычислений агрегатов, например, когда агрегируемые данные имеют значение NULL, и обеспечивают существенные улучшения в производительности кубов, таким образом, давая пользователям возможность увеличения глубины иерархий и сложности вычислений.
- **Обратная запись в MOLAP.** Analysis Services 2008 не требуют выполнения запросов к разделам ROLAP при выполнении обратной записи (writeback), что приводит к существенному повышению производительности таких операций.
- **Масштабируемость Analysis Services.** Одна копия базы данных Analysis Services в режиме «только чтение» может использоваться несколькими серверами Analysis Servers через виртуальный IP-адрес, что позволяет создавать высоко-масштабируемые решения.
- **Сохранение планов выполнения.** В SQL Server 2008 имеется возможность сохранения планов выполнения после рестарта и обновления сервера.

Включенные в состав SQL Server 2008 средства создания отчетов — Reporting Services — позволяют решить одну из самых важных задач работы с данными — предоставление информации в нужный момент именно тем, кто нуждается в ней. Масштабируемость и расширяемая архитектура сервисов отчетов позволяет легко интегрировать отчеты в различные приложения и сервисы.

Поддерживается возможность генерации отчетов на основе данных, полученных из различных источников, включая SQL Server, DB2 и Oracle без необходимости в построении централизованного хранилища данных.



MOLAP Writeback

Возможность создания и управления BI-решениями любой сложности

В организациях, где требуется реализация комплексных аналитических решений, должны быть соответствующие ресурсы для создания и развертывания эффективных BI-решений. В состав SQL Server 2008 входит набор средств, позволяющих упростить разработку и сопровождение аналитических решений.

Интеграция SQL Server 2008 с Visual Studio позволяет разработчикам создавать надежные, безопасные и масштабируемые решения, работающие с данными. Специальное средство разработки – SQL Server Business Intelligence Development Studio представляет собой унифицированную среду для создания различных типов BI-решений, включая ETL, OLAP и отчеты и позволяет работать с такими сервисами, как Integration Services, Analysis Services и Reporting Services. Различные «мастера» существенно облегчают создание BI-приложений и снижают время, требуемое на освоение новых технологий. Поддержка Common Language Runtime на уровне ядра базы данных позволяет разработчикам выбирать наиболее подходящие для задач языки программирования, включая T-SQL, VB, C# и т. п..

К основным возможностям SQL Server 2008, предназначенным для разработчиков BI-решений отнесем:

- SQL Server Business Intelligence Development Studio – единая среда разработки BI-решений, включая Analysis Services OLAP и приложения класса Data Mining;

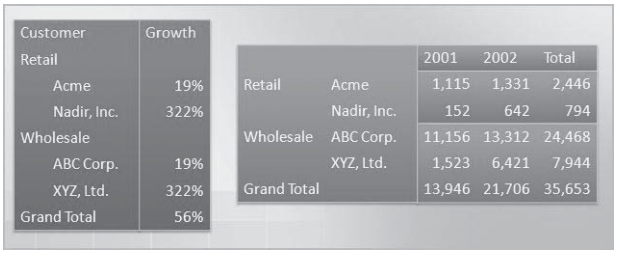
- Так как Business Intelligence Development Studio базируется на Visual Studio, разработчикам доступна поддержка всего цикла разработки, включая дизайн, сборку, отладку и завертывание. Также поддерживается командная разработка с использованием интегрированных средств контроля исходных файлов;
- Набор интуитивных «мастеров» и дизайнеров существенно облегчает создание решений с использованием Analysis Services;
- Дизайнер взаимоотношений между атрибутами позволяет оптимизировать дизайн измерений;
- Редактор измерений автоматически определяет взаимоотношения между родительскими и дочерними измерениями;
- Дизайнер кубов пополнился улучшенными средствами определения и классификации атрибутов , а также идентификации свойств членов;
- Агрегация отдельных разделов позволяет оптимизировать меры на основе различных периодов;
- Дизайнер агрегаций поддерживает новый алгоритм для создания начальных агрегатов. Этот дизайнер оптимизирован для работы с различными типами агрегатов и поддерживает добавление и удаление агрегатов.

Помимо этого, в SQL Server 2008 поддерживается более 40 различных проверок соответствия кода, использующего Analysis Management Objects (AMO), рекомендованным практикам.

Подсистема отчетов является существенным элементом любого BI-решения. В состав SQL Server Reporting Services входит ряд функций, которые облегчают создание подсистем отчетов. К ним относятся:

- Основанная на Visual Studio среда разработки отчетов, которая является частью Business Intelligence Development Studio и используется для создания, тестирования и развертывания отчетов;
- Средство разработки отчетов — Report Builder, предназначенное для конечных пользователей;
- Большой набор структур для отображения данных, включая таблицы, матрицы, списки и графики.

Помимо этого, в состав SQL Server 2008 входят ряд расширений Reporting Services, которые повышают производительность подсистемы отчетов и обеспечивают гибкость при их форматировании и публикации. Одно из расширений в Reporting Services в SQL Server 2008 — это поддержка новой структуры расположения данных, которая объединяет табличные и матричные регионы и называется Tablix. Использование данной структуры позволяет разработчикам создавать отчеты, в которых есть и фиксированные и динамические данные, как это показано на следующем рисунке.



Customer	Growth
Retail	
Acme	19%
Nadir, Inc.	322%
Wholesale	
ABC Corp.	19%
XYZ, Ltd.	322%
Grand Total	56%

		2001	2002	Total
Retail	Acme	1,115	1,331	2,446
	Nadir, Inc.	152	642	794
Wholesale	ABC Corp.	11,156	13,312	24,468
	XYZ, Ltd.	1,523	6,421	7,944
Grand Total		13,946	21,706	35,653

Использование Tablix

Использование единого, унифицированного средства для управления Analysis Services, Reporting Services и Integration Services — SQL Server Management Studio позволяет существенно упростить задачи администрирования и сопровождения BI-решений на базе SQL Server 2008, а средство под названием Performance Studio позволяет централизованно отслеживать использование ресурсов в различных приложениях, работающих с базами данных.

Повышенная масштабируемость и доступность

Новая версия Microsoft SQL Server — SQL Server 2008 позволяет создавать BI-решения, которые могут масштабироваться для поддержки тысяч пользователей, предоставляя им доступ к бизнес-данным компании через знакомый интерфейс на основе Microsoft Office. Продукты семейства Microsoft Office стали полноценным средством обеспечения продуктивности сотрудников, работающих с информацией (information workers) благодаря тесной интеграции с SQL Server 2008. Такая интеграция особенно заметна в новом продукте — Microsoft Office PerformancePoint Server 2007, который позволяет компаниям существенно сократить затраты на получение аналитической информации, связанной, в первую очередь, с финансовыми показателями деятельности компании.

Средства создания отчетов — Reporting Services 2008 поддерживают отрисовку отчетов в Microsoft Office Excel и в Microsoft Office версии с 2000 по 2007. После того как отчеты отрисованы, их можно редактировать средствами соответствующего приложения семейства Office, что существенно расширяет подсистему отчетов, давая пользователям возможность создавать различные документы на основе бизнес-данных, содержащихся в отчетах.

Продукт Microsoft Excel уже давно стал средством для выполнения бизнес- и финансового анализа. Объединение Excel и SQL Server Analysis Services позволяет бизнес-пользователям использовать OLAP-решения через сводные таблицы и графики, включаемые в электронные таблицы Excel. Помимо этого, можно использовать компоненты SQL Server Data Mining Add Ins for Excel 2007 для выполнения различных задач класса «Data Mining», включая подготовку данных, создание, моделирование и управление моделями, а также получение предсказуемых результатов, используя либо

данные, включенные в состав электронных таблиц, либо доступные через базу данных Analysis Services.

Тесная интеграция SQL Server 2008 с Microsoft Office SharePoint Services существенно облегчает публикацию и централизованное управление отчетами на сайтах SharePoint и позволяет создавать различные представления на основе данных, извлекаемых из опубликованных отчетов.

Помимо этого, Microsoft Office PerformancePoint Server предоставляет централизованный интерфейс для анализа данных на основе SQL Server Analysis Services и позволяет пользователям отслеживать, анализировать и планировать их бизнес.

В состав SQL Server 2008 входит ряд расширений, предназначенных для генерации отчетов, которые позволяют быстро и просто создавать различные типы отчетов, отвечающих потребностям компаний, в форматах, которые распространены в той или иной компании и с расположением данных таким образом, чтобы ими можно было пользоваться с максимальной отдачей.

Компонент под названием Report Builder, который существенно расширен в SQL Server 2008, позволяет пользователям простым способом создавать отчеты практически любой структуры. Интуитивный интерфейс дизайнера позволяет даже тем пользователям, которые не имеют навыков в разработке приложений, создавать бизнес-документы на основе данных, содержащихся в отчетах.

Богатые возможности по форматированию позволяют сделать бизнес-документы и отчеты более понятными и интуитивными. Компонент поддержки отформатированного текста (rich text component) позволяет использовать текстовые поля с форматированием и импортировать текстовые строки, а поддержка новых типов графиков и регионов данных типа Tablix позволяет пользователям генерировать отчеты, отвечающие высоким стандартам визуального дизайна.

Новая версия подсистемы отчетов, включенная в состав SQL Server 2008, обладает большей производительностью и масштабируемостью и позволяет публиковать отчеты в Интернете, что делает их доступными для клиентов и поставщиков.

Ключевые новинки в средствах анализа данных

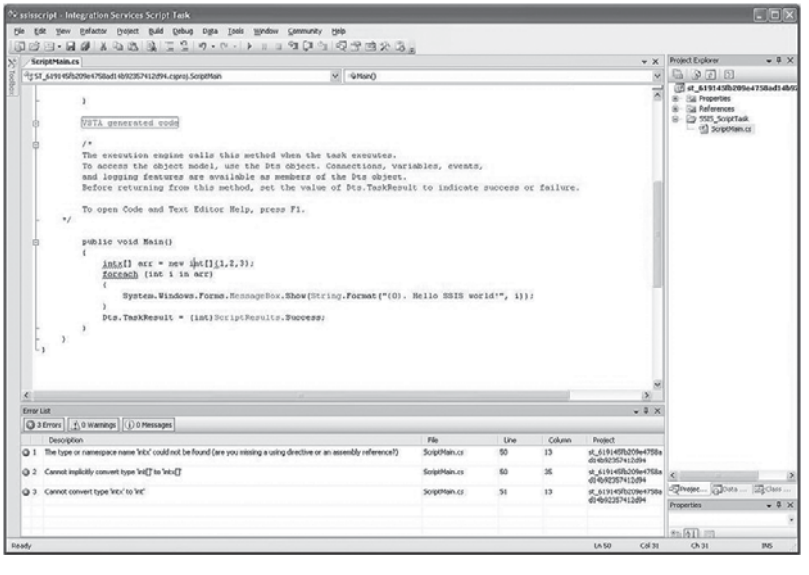
Завершим наше обсуждение средств анализа данных, включенных в состав Microsoft SQL Server 2008, кратким обзором ключевых новинок в основных сервисах — интеграционных сервисах, аналитических сервисах и сервисах отчетов.

Ключевые новинки в SQL Server Integration Services

Как мы отметили выше, SQL Server Integration Services представляют собой платформу для создания высокопроизводительных решений для интеграции данных, включая такие операции, как извлечение данных, их преобразование и загрузка. В состав Integration Services входят графические средства и «мастера», позволяющие создавать и отлаживать трансформационные пакеты, набор задач для выполнения последовательностей действий, средства выполения SQL-команд, отсылки электронных сообщений, поддержку различных источников данных, набор преобразований для очистки, агрегации, объединения и копирования данных, средства управления, сервис Integration Services, используемый для администрирования пакетов, а также программные интерфейсы для доступа к объектной модели Integration Services.

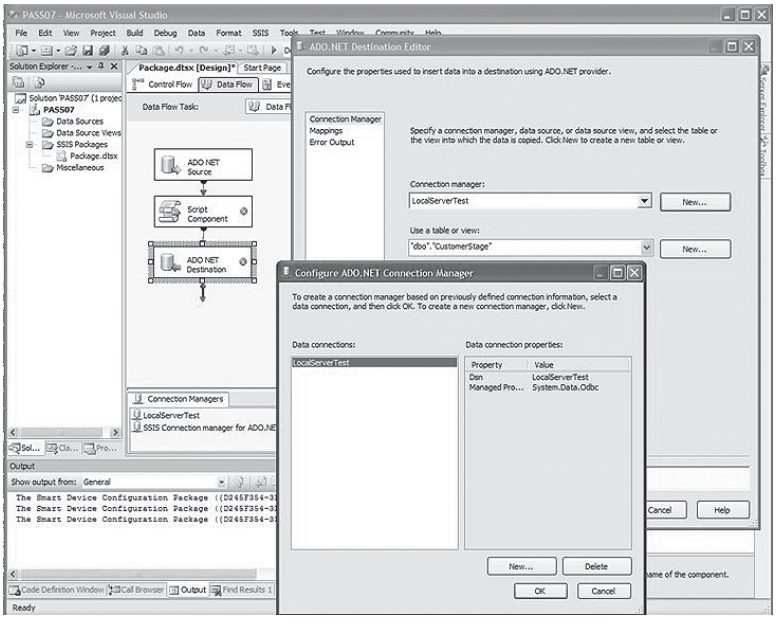
К ключевым новинкам в SQL Server 2008 Integration Services отнесем следующие:

- Поддержка C# Scripting и использование VSTA вместо VSA
  - В SQL Server 2005 для дизайна и выполнения скриптовых программ использовался компонент под названием VSA, который обладал ограниченным по функциональности дизайнером и поддерживал только язык Visual Basic;
  - Использование в версии 2008 новой среды — Visual Studio Tools for Applications (VSTA) позволяет не только улучшить производительность разработчиков, но и обеспечивает поддержку языка C# и возможность включения в код любых .net-сборок, помимо этого поддерживается миграция существующих сценариев и преобразований;



VSTA в SQL Server 2008 Integration Services

- Масштабируемость Pipeline
  - В SQL Server 2008 Integration Services появился новый диспетчер потоков (DataFlow Thread Scheduler), который позволяет оптимизировать длинные цепочки синхронных преобразований.
- Новые компоненты для ADO.Net
  - В состав Integration Services входят следующие компоненты для ADO.Net: **ADO NET Source** (ранее назывался DataReader Source), который потребляет данные, предоставляемые провайдером .NET Framework и делает эти данные доступными для **data flow**, и **ADO NET Destination,** который загружает данные в различные базы данных, совместимые с ADO.NET;
  - Оба компонента используют для связи с данными ADO.NET Connection Manager, который может настраиваться либо через SQL Server Integration Services Designer, либо программно;



SQL Server 2008 Integration Services

- К расширенным механизмам обеспечения поддержки отнесем утилиту SuperDump, которая базируется на технологии Watson и позволяет определить текущее состояние выполняемых пакетов. В случае сбоев статистика собирается в системном протоколе.
- Новый механизм Lookup с поддержкой кэша.
- Ряд новинок в ядре SQL Server также может использоваться в Integration Services:

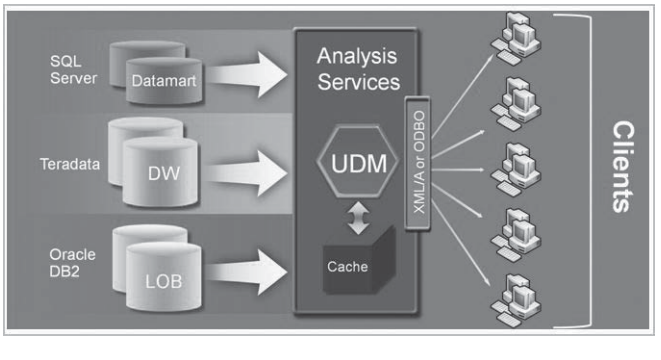
- Новая команда **MERGE**, выполняющая либо UPDATE и INSERT, либо DELETE, в зависимости от результата нахождения соответствия ключей;
- Change Data Capture — позволяет находить все изменения в таблице или наборе таблиц, полностью интегрированный в SQL Server механизм, основанный на журналах, а не на триггерах.

Ключевые новинки в SQL Server Analysis Services

Средства анализа — SQL Server Analysis Services — поддерживают два ключевых подхода к анализу данных — использование многомерных данных и средства Data Mining.

Использование многомерных данных (Multidimensional Data) позволяет создавать многомерные структуры, которые содержат детали и агрегированные данные, полученные из различных источников, таких как реляционные базы данных, в виде унифицированной логической модели, поддерживающей встроенные вычисления. В состав SQL Server Analysis Services входят средства для быстрого, интуитивного, нисходящего анализа больших объемов данных, собранных в унифицированную модель. Механизмы поддержки многомерных данных могут использоваться с хранилищами данных, витринами (Data Mart), базами данных и хранилищами операционных данных и поддерживают работу как с историческими, так и актуальными данными.

Средства Data Mining включают набор стандартных алгоритмов для обработки данных, дизайнер Data Mining Designer, который может использоваться для создания и исследования моделей, а также выполнения анализа и прогнозирования на основе данных, находящихся в моделях. Язык Data Mining Extensions (DMX) может использоваться для управления моделями и создания запросов любой сложности. Все перечисленные выше средства могут использоваться для обнаружения трендов и паттернов, которые могут существовать в массивах данных и, после их обнаружения, использовать эти данные при принятии бизнес-решений.



SQL Server Analysis Services

К новинкам в SQL Server 2008 Analysis Services в области поддержки многомерных данных отнесем:

- Улучшения в дизайне агрегатов:
  - Новый дизайнер агрегатов;
  - Новые «мастера» Aggregation Design и Usage-Based Optimization;
  - Новые механизмы предупреждений – AMO Warnings, срабатывающие в тех случаях, когда пользователи отклоняются от рекомендованных практик дизайна агрегатов.
- Улучшения в дизайне кубов — упрощенный и расширенный дизайнер позволяет создавать более оптимизированные кубы за меньшее число шагов.
- Улучшения в дизайнере размерностей
  - Новый дизайнер Attribute Relationship, включенный в состав редактора размерностей и существенно облегчающий просмотр и изменение взаимоотношений атрибутов;
  - Новые механизмы предупреждений – AMO Warnings, срабатывающие в тех случаях, когда пользователи отклоняются от рекомендованных практик дизайна размерностей или делают логические ошибки в дизайне базы данных;
  - Улучшенный и расширенный мастер Dimension Wizard. В новой версии мастер автоматически определяет иерархии, обеспечивает более защищенные от ошибок конфигурации и позволяет задавать свойства членов размерностей;
  - Новая диалоговая панель Key Columns, облегчающая редактирование ключевых колонок. Помимо этого, такие колонки поддерживаются и в панели Properties;
  - Обновленная вкладка Dimension Structure, которая теперь взаимодействует с новым дизайнером Attribute Relationship и упрощает модификацию атрибутов и иерархий.
- Улучшения в создании резервных копий
  - Используются новые структуры хранения практически неограниченного размера, которые обеспечивают более надежное архивирование баз данных, и реализована расширенная производительность для всех сценариев создания резервных копий и восстановления данных.

К ключевым новинкам в SQL Server 2008 Analysis Services в области Data Mining отнесем:

- В новой версии Analysis Services появилось множество новых методов для обзора моделей данных. Помимо этого, в новой версии включены улучшения в структуры данных, возможности фильтрации внешних наборов данных (на уровне редакторов фильтров в Business Intelligence Development Studio), моделей, расширенные запросы к структурам дан-

ных и возможность доступа к данным на уровне структур непосредственно из моделей.

Ключевые новинки в SQL Server Reporting Services

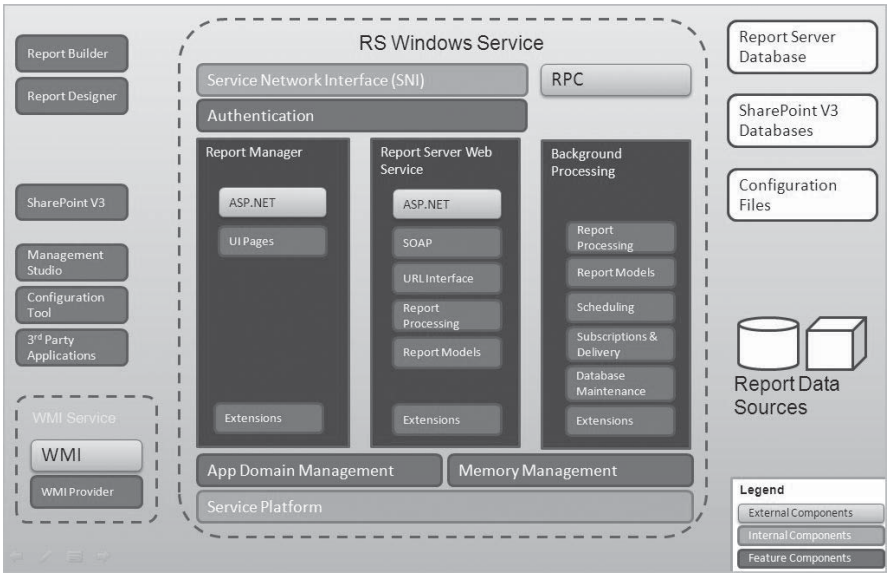
SQL Server 2008 Reporting Services — это серверная платформа для создания комплексных отчетов на основе данных, полученных из различных источников. В состав Reporting Services включены различные средства для создания и управления отчетами, доставки отчетов, а также программные интерфейсы, позволяющие разработчикам интегрировать отчеты в собственные приложения.

Средства Reporting Services работают в среде Microsoft Visual Studio и полностью интегрированы с соответствующими средствами и сервисами SQL Server.

С помощью Reporting Services можно создавать интерактивные, табличные, графические отчеты и отчеты свободной формы на основе реляционных, многомерных или XML-данных, получаемых из различных источников. Поддерживается возможность публикации отчетов, задания обработки и генерации отчетов по расписанию или доступ к отчетам по запросу. Средства Reporting Services также позволяют создавать отчеты на основе предопределенных моделей и интерактивно исследовать данные в рамках модели. Поддерживается множество форматов просмотра, экспорт отчетов в другие приложения и подписка на опубликованные отчеты. Созданные отчеты могут просматриваться в веб-браузере, как часть Windows-приложения или на сайте SharePoint.

К ключевым новинкам в SQL Server 2008 Reporting Services отнесем:

- Новая архитектура Report Server 2008
  - Report Server не хостится в Internet Information Services, а напрямую использует HTTP.SYS и сетевой стек SQL Server (SQL OS, SQL CLR и SQL Network Interface).
- Масштабируемость ядра Report Engine
  - В SQL Server 2005 отчеты располагались в памяти и, соответственно, объем используемой памяти был пропорционален объему данных в отчете. Большие объемы данных приводили к сбоям и ошибкам и сложностям с поддержкой в экспорте в такие форматы, как PDF, Excel и CSV;
  - В SQL Server 2008 ядро подсистемы отчетов выполняет обработку отчетов по запросу и использует иерархическую объектную модуль на основе курсоров. Помимо этого реализована новая архитектура отрисовки отчетов;

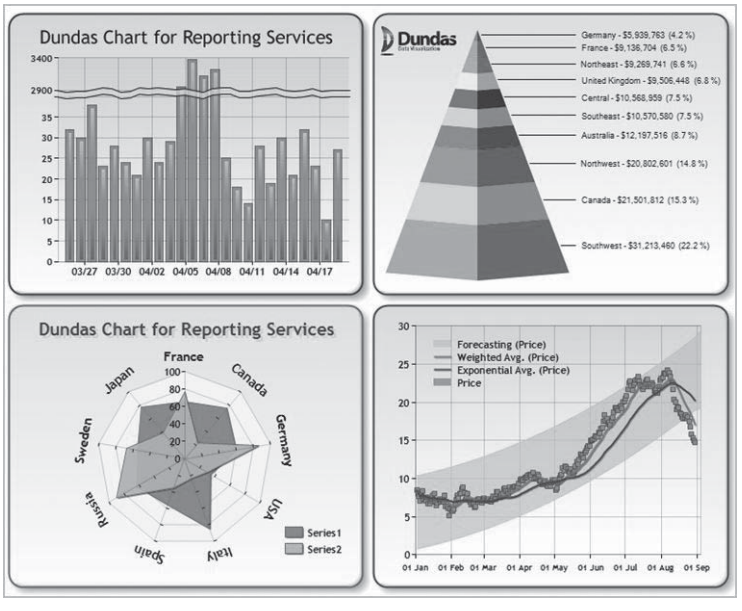


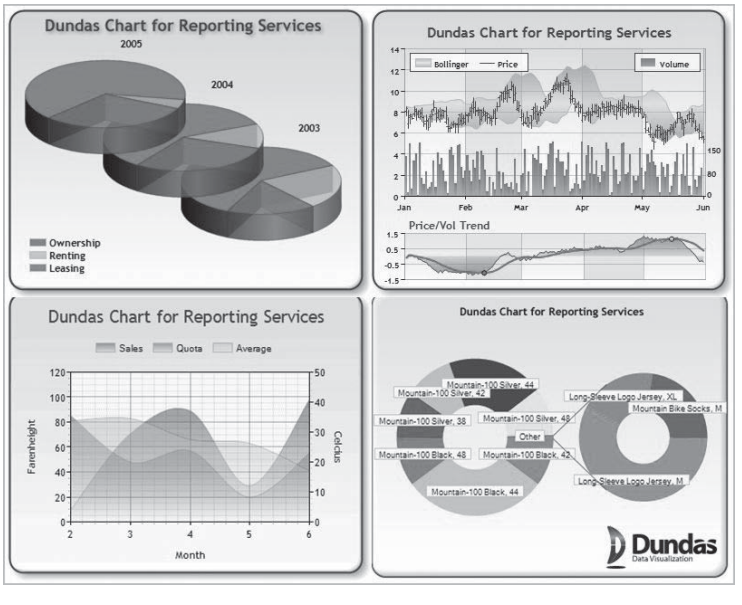
Архитектура сервера Report Server 2008

- Перечисленные выше изменения привели к более синхронизованным действиям по расположению элементов отчетов и разделении отчетов на страницы, поддержке распределенной архитектуры клиент/сервер, более надежной отрисовке, использующей возможности клиентских компьютеров, улучшениям в скорости отображения первой страницы отчета и возможностям реализации расположения элементов в виде Tablix.

■ Использование компонентов Dundas Software Data Visualization

- В SQL Server 2008 поддерживаются следующие компоненты:
  - Chart;
  - Gauge;
  - Map;
  - Barcode;
  - Calendar.





Компоненты Dundas Software Data Visualization

В следующем разделе мы кратко рассмотрим основные ресурсы для дальнейшего изучения Microsoft SQL Server 2008.

Приложение. Ключевые ресурсы по SQL Server 2008

Основные ресурсы

Основная страница продукта на сайте Microsoft

- www.microsoft.com/sql/2008/default.mspx

Видео-обзор продукта

- http://www.microsoft.com/sql/2008/all_up/index.html

Демонстрация возможностей продукта

- http://go.microsoft.com/?linkid=8075816

Веб-трансляции по продукту

- http://www.microsoft.com/sql/2008/learning/webcasts.mspx

Аудио-трансляции по продукту

- http://www.microsoft.com/events/series/technetsqlserver2008.aspx?tab=podcasts

Виртуальные лабораторные работы

- http://www.microsoft.com/events/series/technetsqlserver2008.aspx?tab=virtuallabs

Технические ресурсы

- http://msdn2.microsoft.com/en-us/sql/bb498264.aspx

Обзорные документы по ключевым возможностям продукта

- http://www.microsoft.com/sql/2008/learning/whitepapers.mspx
- SQL Server 2008 Product Overview
- http://www.microsoft.com/sql/techinfo/whitepapers/sql2008overview.mspx
- **Видео —** SQL Server 2008 Overview Demo
- http://go.microsoft.com/?linkid=8075816
- **Видео —** Microsoft SQL Server Data Platform Vision — SQL Server Expert Series
- http://www.microsoft.com/sql/2008/all_up/index.html
- **Видео —** SQL Server 2008 Overview — Microsoft Business Intelligence Summit (May 2007)
- http://www.microsoft.com/sql/2008/bisummit.asx

Документация — SQL Server 2008 Books Online

- http://msdn2.microsoft.com/library/bb543165.aspx

Использование реляционных данных (OLTP)

- http://www.microsoft.com/sql/2008/solutions/oltp.mspx
- OLTP in SQL Server 2008

- http://www.microsoft.com/sql/techinfo/whitepapers/hosting_sql_2008_oltp.mspx
- SQL Server 2008 Security Overview for Database Administrators
- http://www.microsoft.com/sql/techinfo/whitepapers/SQL2008_Sec_Overview.mspx
- SQL Server 2008 Manageability
- http://www.microsoft.com/sql/techinfo/whitepapers/sql_2008_manageability.mspx
- SQL Server 2008 Performance and Scale
- http://www.microsoft.com/sql/techinfo/whitepapers/sql_2008_perfandscale.mspx

Разработка приложений

- http://www.microsoft.com/sql/2008/solutions/developer.mspx
- Data Platform Development
- http://www.microsoft.com/sql/techinfo/whitepapers/hosting_sql_2008_dp.mspx
- What's New for XML in SQL Server 2008?
- http://www.microsoft.com/sql/techinfo/whitepapers/sql_2008_xml.mspx Managing Unstructured Data with SQL Server 2008
- http://www.microsoft.com/sql/techinfo/whitepapers/sql_2008_unstructured.mspx
- SQL Server 2008: Delivering Location Intelligence with Spatial Data
- http://www.microsoft.com/sql/techinfo/whitepapers/spatialdata.mspx
- **Видео —** SQL Server 2008 Beyond Relational – SQL Server Expert Series
- http://www.microsoft.com/sql/2008/beyond/index.html
- **Видео —** SQL Server 2008 ADO.NET – SQL Server Expert Series
- http://www.microsoft.com/sql/2008/entity/index.html
- **Видео —** SQL Server 2008 Dynamic Development Enhancements — SQL Server Expert Series
- http://www.microsoft.com/sql/2008/dynamic/index.html

Консолидация серверов

- http://www.microsoft.com/sql/2008/solutions/consolidate.mspx
- Server Consolidation with SQL Server 2008
- http://www.microsoft.com/sql/techinfo/whitepapers/hosting_sql_srv_consol.mspx
- **Видео —** SQL Server 2008 Enterprise Data Platform Vision — SQL Server Expert Series
- http://www.microsoft.com/sql/2008/edp/index.html

Аналитика — Business Intelligence и Data Warehosing

- http://www.microsoft.com/sql/2008/solutions/bi.mspx
- http://www.microsoft.com/sql/2008/solutions/datawarehouse.mspx
- Business Intelligence with SQL Server 2008
- http://www.microsoft.com/sql/techinfo/whitepapers/hosting_sql_2008_bi.mspx
- SQL Server 2008 Analysis Services Overview
- http://www.microsoft.com/sql/techinfo/whitepapers/sql2008ASOverview.mspx
- Predictive Analysis with SQL Server 2008
- http://www.microsoft.com/sql/techinfo/whitepapers/SQL2008PredictAnalysis.mspx
- Business Intelligence in SQL Server 2008
- http://www.microsoft.com/sql/techinfo/whitepapers/hosting_sql_2008_bi.mspx
- SQL Server 2008 Reporting Services Whitepaper
- http://www.microsoft.com/sql/techinfo/whitepapers/sql_2008_ssrs.mspx
- An Introduction to SQL Server 2008 Integration Services
- http://www.microsoft.com/sql/techinfo/whitepapers/SSIS2008_Intro.mspx
- New Datawarehouse Scalability Features
- http://www.microsoft.com/sql/techinfo/whitepapers/SQL2008IntroDW.mspx
- **Видео** — Microsoft SQL Server 2008 Pervasive Insight Vision — SQL Server Expert Series
- http://www.microsoft.com/sql/2008/bi/index.html
- **Видео** — Microsoft SQL Server 2008 Data Mining Enhancements — SQL Server Expert Series
- http://www.microsoft.com/sql/2008/datamining/index.html
- **Видео** — SQL Server 2008 Data Warehousing Vision — SQL Server Expert Series
- http://www.microsoft.com/sql/2008/warehousing/index.html

Об авторе

Автор выражает благодарность
Алексею Шуленину и Андрею Хромову
за полезные комментарии
в процессе написания данной книги

Алексей Федоров осваивал азы компьютерной грамотности на СМ-4 (клон PDP-11, созданный странами соцлагеря) и ЕС-1840/41 (советский клон IBM PC). В зрелые годы занимался русификацией принтеров Epson FX-80, локализацией операционной системы DR-DOS (компании Digital Research) и средствами разработки Borland C++, читал лекции, писал статьи для российских и международных компьютерных изданий (более 400 статей), книги по различным вопросам программирования (на русском и английском языках). Работал руководителем Управления разработки программного обеспечения, техническим директором швейцарской веб-компании.

В Microsoft — 5 лет, в настоящее время возглавляет отдел по работе с партнерами в Департаменте стратегических технологий. В 2007 г. был награжден как победитель конкурса «Лучший по профессии».

Алексей Федоров

Microsoft® SQL Server™ 2008.

Краткий обзор ключевых новинок

Алексей Федоров — сотрудник отдела стратегических технологий
ООО «Майкрософт Рус» (alexeif@microsoft.com)

Подготовлено к печати издательством «Русская Редакция»
123290, Москва, Шелепихинская наб., д. 32
тел.: (495) 256-6691, тел./факс: (495) 256-7145
e-mail: info@rusedit.com, http://www.rusedit.com

РУССКАЯ РЕДАКЦИЯ