**министерство образования и науки Российской Федерации**

Старооскольский технологический институт им. А.А. УГАРОВА

(филиал) федерального государственного автономного образовательного учреждения

высшего образования

«Национальный исследовательский технологический университет «МИСиС»

**ОСКОЛЬСКИЙ ПОЛИТЕХНИЧЕСКИЙ КОЛЛЕДЖ**

УТВЕРЖДено

НМС опк

пРОТОКОЛ №1

ОТ «01» сентября 2016 г.

**ОСНОВЫ ПРОЕКТИРОВАНИЯ БАЗ ДАННЫХ**

***Методические указания для студентов очной формы обучения для выполнения самостоятельной работы***

Специальность 09.02.04 Информационные системы (по отраслям)

Старый Оскол 2016г

|  |  |
| --- | --- |
| *Рассмотрены на заседании П(Ц)К 09.02.04*  *Протокол №*  *от «\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_ 2016г.*  *Председатель*  *Назарова О.И.* | *Методические указания составлены в соответствии с рабочей программой по дисциплине*  *Основы проектирования баз данных*  *Специальности*  *09.02.04 Информационные системы (по отраслям)*  *Зам .директора по М Р*  *к.п.н., доцент \_\_\_\_\_\_\_\_\_\_\_\_\_А.М. Степанова* |

***Составитель: Артюхина Д.Д., Коренькова Т.Н.***

***Рецензенты:***

*внутренний: Горюнова М.В.* - преподаватель ОПК СТИ НИТУ МИСиС

*внешний:* Анпилов А.Э. – инженер ООО «КМАЭМ»

**СОДЕРЖАНИЕ**

**ВВЕДЕНИЕ …………………………………………………………………………………….4**

**ВЫПОЛНЕНИЕ ПРОЕКТА №1 …………………………………………………………….5**

**ВЫПОЛНЕНИЕ ПРОЕКТА №1 ……………………………………………………………43**

**СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ ……………………………………...78**

ВВЕДЕНИЕ

**Структурированный язык запросов SQL** (Structured Query Language) - это язык, разработанный корпорацией IBM в 1970 году. Он фактически стал стандартом в качестве языка [реляционных баз данных](file:///C:\Users\Дарья\Desktop\xBase\УЧЕБНИК%20Xbase,%20SQL\УЧЕБНИК%20Xbase,%20SQL\FILES\Глоссарий.html#RSB). Cлужащий IBM доктор Е.Ф.Кодд предложил язык SQL (называвшийся тогда SEQUEL - Structured English Query Language, структурированный английский язык запросов), как средство извлечения информации из реляционной базы данных, модель которой он разработал в 1970 г.

В настоящее время SQL представляет собой не просто язык запросов, а наиболее распространенный язык управления реляционными базами данных типа [клиент-сервер](file:///C:\Users\Дарья\Desktop\xBase\УЧЕБНИК%20Xbase,%20SQL\УЧЕБНИК%20Xbase,%20SQL\FILES\Глоссарий.html#KS). Основное достоинство SQL заключается в том, что он унифицирован: стандартный набор инструкций SQL можно использовать в любой [системе управления базами данных](file:///C:\Users\Дарья\Desktop\xBase\УЧЕБНИК%20Xbase,%20SQL\УЧЕБНИК%20Xbase,%20SQL\FILES\Глоссарий.html#subd), которая поддерживает SQL. Именно язык SQL является стандартом для работы с реляционными СУБД. SQL стал единственным языком баз данных клиент-сервер. [Сервер баз данных](file:///C:\Users\Дарья\Desktop\xBase\УЧЕБНИК%20Xbase,%20SQL\УЧЕБНИК%20Xbase,%20SQL\FILES\глоссарий.html#sbd) (нижний уровень) отвечает за хранение данных. [Приложения-клиенты](file:///C:\Users\Дарья\Desktop\xBase\УЧЕБНИК%20Xbase,%20SQL\УЧЕБНИК%20Xbase,%20SQL\FILES\глоссарий.html#pk) (верхний уровень) добавляют или обновляют данные. Кроме того, приложение генерирует инструкции SQL. При регулярной работе с базами данных знание SQL обязательно. Это также относится к разработчикам, которым требуется создавать приложения с определяемыми пользователем запросами.

Создание современных информационных систем представляет собой сложнейшую задачу, решение которой требует применения специальных методик и инструментов. Неудивительно, что в последнее время среди системных аналитиков и разработчиков значительно вырос интерес к CASE (Computer-Aided Software/System Engineering) - технологиям и инструментальным CASE-средствам, позволяющим максимально систематизировать и автоматизировать все этапы разработки программного обеспечения.

Основной целью самостоятельной работы по дисциплине «Основы проектирования баз данных» является получение практических навыков по созданию и модификации структур таблиц данных и БД; загрузке и модификации данных; работе с файлами; индексированию и поиску данных в БД; программированию в СУБД; организации меню; созданию запросов на языке SQL, различных типов отчетов, форм с использованием мастеров и конструкторов.

ВЫПОЛНЕНИЕ ПРОЕКТА №1

**1. Введение в MS SQL Server**

**1.1 Платформа SQL Server**

SQL Server – это комплексная платформа баз данных, обеспечивающая управление данными в масштабе предприятия и оснащённая интегрированными средствами бизнес-аналитики (Business Intelligence, BI). Ядро СУБД SQL Server обеспечивает безопасное и надёжное хранение данных в реляционном формате, в формате XML а так же объектов среды CLR Microsoft .NET Framework. Благодаря поддержке широкого набора форматов, SQL Server обеспечивает гибкость, необходимую для создания современных комплексных решений. Эта платформа также предоставляет возможность создания и управления высокопроизводительными приложениями баз данных с высоким уровнем доступности.

В основе решения управления корпоративными данными лелажит ядро базы данных SQL Server. Помимо поддержки реляционных баз данных и данных в формате XML, SQL Server также сочетает лучшее в таких областях, как анализ, подготовка отчетов, интеграция данных и рассылка уведомлений. Выгодным преимуществом SQL Server является тесная интеграция с Microsoft Visual Studio, Microsoft Office System и целым комплектом средств разработки, включая Business Intelligence Development Studio. Это позволяет создавать и развёртывать экономически эффективные решения бизнес-аналитики, которые позволят сотрудникам доставлять аналитическую информацию в любое место предприятия при помощи карт ключевых показателей эффективности, панелей управления, веб-служб и мобильных устройств.

Сервер баз данных Microsoft SQL Server в качестве языка запросов использует версию языка SQL, получившую название Transact-SQL (сокращённо T-SQL). Язык T-SQL является реализацией SQL-92 (стандарт ISO для языка SQL) с множественными расширениями.

В SQL Server начиная с версии 2005 встроена поддержка .NET Framework. Благодаря этому хранимые процедуры БД могут быть написаны на любом языке платформы .NET, используя полный набор библиотек, доступных для .NET Framework, включая Common Type System (система обращения с типами данных в Microsoft .NET Framework). Однако, в отличие от других процессов, .NET Framework, будучи базисной системой для SQL Server 2005, выделяет дополнительную память и выстраивает средства управления SQL Server вместо того, чтобы использовать встроенные средства Windows. Это повышает производительность в сравнении с общими алгоритмами Windows, так как алгоритмы распределения ресурсов специально настроены для использования в структурах SQL Server.

Версия SQL Server 2008 R2 направлена на то, чтобы сделать управление данными самонастраивающимся, самоорганизующимся и самообслуживающимся механизмом – для реализации этих возможностей были созданы технологии SQL Server Always On. Это позволяет уменьшить до нуля время нахождения сервера в нерабочем состоянии.

В SQL Server 2008 для поддержки структурированных и частично структурированных данных, включая цифровые форматы для изображений, звуков, видео и других типов мультимедиа был добавлен новый тип хранения больших двоичных объектов (BLOB) FILESTREAM, который объединяет компонент Database Engine с файловой системой NTFS, размещая данные больших двоичных объектов (BLOB) в файловой системе в виде файлов. Манипулирование данными, хранящимися в FILESTREAM осуществляется при помощи инструкций Transact-SQL, что позволяет использовать оптимизированное хранилище без изменения логики приложений его использующих. Интерфейсы файловой системы Windows также обеспечивают потоковый доступ к этим данным.

Кроме этого, были включены специализированные форматы даты и времени и пространственные (Spatial) типы для пространственно зависимых данных.

Для повышения эффективности администрирования в SQL Server были включены библиотеки Declarative Management Framework, позволяющие назначать в декларативной форме полномочия для баз данных или отдельных объектов БД. Были улучшены методы компрессии данных.

**1.2 Выпуски SQL Server**

Различные выпуски SQL Server удовлетворяют индивидуальным требованиям каждой организации или отдельного лица к производительности, среде выполнения и цене. Набор устанавливаемых компонентов SQL Server зависит от потребностей конкретного пользователя.

В табл. 1.1 приведено описание серверных выпусков SQL Server.

Специализированные выпуски SQL Server предназначены для особых пользовательских сред. В табл. 1.2 описаны эти выпуски SQL Server.

Более подробное сравнение функциональных возможностей различных редакций SQL Server 2008 R2 можно по адресу <http://msdn.microsoft.com/ru-ru/library/cc645993.aspx>.

**1.2.1 Использование SQL Server 2008 с клиент-серверными приложениями.** На компьютер, где работают клиент-серверные приложения, которые подключаются непосредственно к экземпляру SQL Server, достаточно установить только клиентские компоненты SQL Server. Установка клиентских компонентов будет хорошим выбором также и в том случае, если администрируется экземпляр SQL Server на сервере базы данных или планируется разработка приложений SQL Server.

При выборе установки клиентских компонентов будут установлены следующие компоненты SQL Server: программы командной строки, средства служб Reporting Services, компоненты связи, модели программирования, средства управления и разработки и электронная документация по SQL Server.

Таблица 1.1

Описание серверных выпусков SQL Server

|  |  |
| --- | --- |
| **Выпуск** | **Определение** |
| Datacenter  (x86, x64 и IA64) | Построенный на основе SQL Server 2008 R2 Enterprise, редакция SQL Server 2008 R2 Datacenter представляет собой высокопроизводительную платформу обработки данных, предоставляющую самый высокий уровень масштабируемости для крупных приложений, виртуализации и консолидации, а также управления инфраструктурой базы данных организации, что позволяет организациям эффективно определять ответственную среду. |
| Enterprise  (x86, x64 и IA64) | Выпуск SQL Server 2008 R2 Enterprise является многофункциональной платформой обработки данных, обеспечивающей встроенную безопасность, доступность и масштабируемость в сочетании с надежными возможностями бизнес-аналитики, что позволяет добиться высокого уровня обслуживания даже при критических рабочих нагрузках. |
| Standard  (x86 и x64) | Выпуск SQL Server 2008 R2 Standard является идеальной платформой для обработки данных и бизнес-аналитики, необходимой отделам и малым организациям для выполнения их приложений, что позволяет эффективно управлять базами данных при минимальном использовании ИТ-ресурсов.  Выпуск SQL Server Standard for Small Business включает все технические компоненты и возможности выпуска SQL Server Standard и лицензирован для использования в компаниях малого бизнеса, число компьютеров в которых не превышает 75. |

Таблица 1.2

Описание выпусков SQL Server

|  |  |
| --- | --- |
| **Условие** | **Определение** |
| 1 | 2 |
| SQL Server Developer  (x86, x64 и IA64) | Выпуск SQL Server Developer позволяет разработчикам строить приложения любого вида на базе SQL Server. Этот выпуск включает все функциональные возможности выпуска SQL ServerDatacenter, однако лицензируется для разработки и тестирования системы, а не для применения в качестве рабочего сервера. Выпуск SQL Server Developer является идеальным выбором для тех, кто создает и тестирует приложения. Выпуск SQL Server Developer можно обновить для производственного использования. |
| SQL Server Workgroup  (x86 и x64) | Выпуск SQL Server Workgroup является идеальным выбором для баз данных отделений - надежная платформа для управления данными и создания отчетов, включающая возможности обеспечения безопасности, удаленной синхронизации и управления. |
| SQL Server Web (x86 и x64) | Выпуск SQL Server Web - это вариант с низкой общей стоимостью владения, предназначенный для размещения веб-сайтов, который обеспечивает масштабируемость и функции управления для небольших и крупномасштабных веб-свойств. |
| SQL Server Express (x86 и x64)  SQL Server Express with Tools  (x86 и x64)  Выпуск SQL Server Express with Advanced Services (x86 и x64) | Платформа баз данных SQL Server Express основана на SQL Server. Она также является заменой для Microsoft Desktop Engine (выпуск MSDE). Благодаря интеграции со средой Visual Studio, выпуск SQL Server Express облегчает разработку управляемых данными приложений, обладающих богатыми возможностями, обеспечивающими безопасность хранения данных и не требующими много времени для развертывания.  SQL Server Express - бесплатный выпуск, который могут распространять независимые поставщики ПО (при соблюдении соответствующего соглашения). Выпуск SQL Server Express идеально подходит для обучения, а также для построения настольных или небольших серверных приложений. Этот выпуск - лучший выбор для независимых поставщиков программного обеспечения, непрофессиональных разработчиков и любителей, которые хотят создавать клиентские приложения. Если необходимы дополнительные функции базы данных, SQL Server Express можно с легкостью обновить до расширенных версий SQL Server. |

Окончание таблицы 1.2

|  |  |
| --- | --- |
| **1** | **2** |
| Compact 3.1 (x86) | SQL Server Compact 3.5 - бесплатная внедренная база данных, которая идеально подходит для построения изолированных и иногда сетевых приложений для мобильных устройств, рабочих станций и веб-клиентов, работающих на всех платформах Windows. |

**1.3 Компоненты SQL Server**

При установке SQL Server мастер установки на странице «Выбор компонентов» позволяет выбрать компоненты SQL Server, которые должны быть установлены (табл. 1.3).

Таблица 1.3

Описание компонентов SQL Server

|  |  |
| --- | --- |
| **Компоненты сервера** | **Описание** |
| SQL Server Database Engine | В состав компонента SQL Server Database Engine входит компонент Database Engine, основная служба для хранения, обработки и обеспечения безопасности данных, репликации, полнотекстового поиска и средств управления реляционными и XML-данными. |
| Analysis Services | Службы Analysis Services содержат средства создания и управления приложениями аналитической обработки в сети (OLAP) и приложениями интеллектуального анализа данных. |
| Reporting Services | Службы Reporting Services включают в себя серверные и клиентские компоненты для создания, управления и развертывания табличных, матричных и графических отчетов, а также отчетов в свободной форме. Службы Reporting Services являются расширяемой платформой, которую можно использовать для разработки приложений отчетов. |
| Integration Services | Службы Integration Services представляют собой набор графических средств и программируемых объектов для перемещения, копирования и преобразования данных. |

В таблице 1.4 представлено описание основных средств управления SQL Server.

Таблица 1.4

Описание средств управления SQL Server

|  |  |
| --- | --- |
| **Средства  управления** | **Описание** |
| SQL Server Management Studio | Среда SQL Server Management Studio представляет собой интегрированную среду для доступа, настройки, управления, администрирования и разработки компонентов SQL Server. Среда Management Studio позволяет работать с SQL Server разработчикам и администраторам любого уровня подготовки. Обозреватель Internet Explorer 6 с пакетом обновления 1 или более поздней версии необходим для любой установки среды Management Studio. |
| Диспетчер конфигурации SQL Server | Диспетчер конфигурации SQL Server обеспечивает базовые возможности управления конфигурациями для служб, серверных протоколов, клиентских протоколов и псевдонимов клиентов SQL Server. |
| SQL Server Profiler | Приложение SQL Server Profiler предоставляет графический пользовательский интерфейс для наблюдения за экземпляром компонента Database Engine или служб Analysis Services. |
| Помощник по настройке ядра СУБД | Помощник по настройке ядра СУБД помогает создавать оптимальные наборы индексов, индексированных представлений и секций. |
| Business Intelligence Development Studio | Среда Business Intelligence Development Studio представляет собой интегрированную среду разработки для решений служб Analysis Services, Reporting Services и Integration Services. Обозреватель Internet Explorer 6 с пакетом обновления 1 или более поздней версии необходим для любой установки среды BI Development Studio. |
| Компоненты связи | Устанавливает компоненты для связи между клиентами и серверами и сетевые библиотеки для DB-библиотеки, ODBC и OLE DB. |

**1.4 Экземпляры SQL Server**

На одном компьютере одновременно можно запустить несколько экземпляров SQL Server, и каждый экземпляр выполняется независимо от других.

Каждый экземпляр SQL Server состоит из отдельного набора служб, которые имеют специальные параметры сортировки и другие настройки уровня сервера. Структура каталогов, структура реестра и имена служб отражают имя и идентификатор конкретного экземпляра, создаваемого в процессе установки SQL Server.

Каждый экземпляр является либо экземпляром по умолчанию, либо именованным. Имя экземпляра по умолчанию – «MSSQLSERVER». Имя экземпляра SQL ServerExpress по умолчанию – «SQLExpress». Клиенту при установлении соединения имя экземпляра указывать необязательно. Именованный экземпляр определяется пользователем при установке и может быть установлен без предварительной установки экземпляра по умолчанию. Экземпляром по умолчанию может быть SQL Server 2000, SQL Server 2005 или SQL Server 2008.

На одном сервере или процессоре поддерживается несколько экземпляров SQL Server, но только один из них независимо от версии может быть экземпляром по умолчанию. Все остальные экземпляры должны быть именованными.

**1.5 Системные базы данных**

Когда Вы устанавливаете SQL Server, автоматически создаётся четыре системные базы данных, которые играют жизненно важную роль для нормального функционирования SQL Server - Master, Resource, Model, Msdb и Tempdb.

**1.5.1 База данных Master.** Эта база данных хранит всю критическую для сервера информацию о конфигурации, учётных записях пользователей, запускаемых процессах, сообщениях об ошибках, расположении главных файлов баз данных и т.д. При её повреждении, можно использовать утилиту rebuildm.exe, которая восстанавливает базу Master и затем восстановить пользовательские базы. Старайтесь всегда иметь актуальную копию системной базы данных Master.

**1.5.2 База данных Resource.** База данных Resource содержит все системные объекты, входящие в SQL Server, и доступна только для чтения. Физические файлы базы данных Resource имеют имена mssqlsystemresource.mdf и mssqlsystemresource.ldf. С каждым экземпляром SQL Server может быть связан один и только один файл mssqlsystemresource.mdf; кроме того, экземпляры не могут использовать этот файл совместно.

Системные объекты SQL Server, такие как sys.objects, физически хранятся в базе данных Resource, а логически отображаются в схеме sys для каждой базы данных. База данных Resource не содержит пользовательских данных или метаданных и скрыта от пользователя, она не отображается в Обозревателе объектов, информацию о ней нельзя получить вызовом системной хранимой процедуры sp\_helpdb или запросом к представлению sys.databases.

Резервное копирование базы данных Resource средствами SQL Server не предусмотрено. Пользователь может создать резервную копию файла mssqlsystemresource.mdf или диска с этим файлом сторонними средствами, но восстановить такие резервные копии с помощью SQL Server не удастся. Восстановить резервную копию файла mssqlsystemresource.mdf можно будет только вручную, при этом следует соблюдать осторожность, чтобы не перезаписать текущую базу данных Resource устаревшей или потенциально небезопасной версией.

**1.5.3 База данных Model.** Эта база данных используется, как шаблон для новых баз данных то есть, всякий раз, когда Вы создаете новую базу данных, объекты в этой базе данных копируется в новую базу данных. Например, когда создается новая база данных <db name> без указания её размера, SQL создаст её с размер Model, то есть 0.75 МБ и скопирует все объекты из образцовой базы в новую базу данных. Если Вы хотите исполнить некоторые схожие действия для всех вновь создаваемых пользовательских базах данных, Вы можете включать необходимые объекты, таблицы, процедуры и т. п. в Model, и тогда, каждая новая база данных также будет их содержать.

**1.5.4 База данных Msdb.** Это одна из важных системных баз данных SQL, обеспечивающая его управление и обслуживание. База данных Msdb содержит задания (task-scheduling), обработку исключений, аварийное управление и информацию об операторах системы, то есть содержит информацию для всех операторов об их адресах электронной почты или номерах пейджера, а также информацию об истории по всем сеансам резервного копирования или восстановления баз данных.

**1.5.5 База данных Tempdb.** Следуя своему названию, эта база данных хранит все временные таблицы и промежуточные результаты group by, order by, distinct и т.д. Временные данные хранятся в Tempdb, пока хоть один пользователь подключён к SQL Server. Существует два вида временных таблиц: локальные и глобальные. Локальные временные таблицы видны только их создателям до завершения сеанса соединения с экземпляром SQL Server, как только они впервые созданы или когда на них появляется ссылка. Локальные временные таблицы удаляются после отключения пользователя от экземпляра SQL Server. Глобальные временные таблицы видны всем пользователям в течение любых сеансов соединения после создания этих таблиц и удаляются, когда все пользователи, ссылающиеся на эти таблицы, отключаются от экземпляра SQL Server. Tempdb создаётся каждый раз заново при запуске SQL сервера, поэтому все временные данные будут потеряны после перезагрузки SQL сервера или при его крахе.

**2. Создание базы данных**

В одном экземпляре SQL Server может быть создано до 32 767 баз данных. В каждой базе данных должно быть по крайней мере 2 файла (первичный файл и файл журнала транзакций) и по крайней мере одна файловая группа. Для каждой базы данных может указываться не более 32 767 файлов и 32 767 файловых групп.

При создании новой базы данных в неё копируются все пользовательские объекты из базы данных **model**. Таким образом, можно добавлять любые объекты, например, таблицы, представления, хранимые процедуры и типы данных, в базу данных **model**, чтобы включать их во все вновь создаваемые базы данных. Файлы данных следует делать как можно большего размера, в соответствии с максимальным предполагаемым объёмом данных в базе данных.

**2.1 Команда создания базы данных**

Создание базы данных осуществляется выполнением команды **CREATE DATABASE**:

CREATE DATABASE database\_name

[ ON

{ [ PRIMARY ] [ <filespec> [ ,...n ]

[ , <filegroup> [ ,...n ] ]

[ LOG ON { <filespec> [ ,...n ] } ] }

]

[ COLLATE collation\_name ]

]

<filespec> ::=

{

(

NAME = logical\_file\_name ,

FILENAME = { 'os\_file\_name' }

[ , SIZE = size [ KB | MB | GB | TB ] ]

[ , MAXSIZE = { max\_size [ KB | MB | GB | TB ] | UNLIMITED } ]

[ , FILEGROWTH = growth\_increment [ KB | MB | GB | TB | % ] ]

) [ ,...n ]

}

<filegroup> ::=

{

FILEGROUP filegroup\_name [ CONTAINS FILESTREAM ] [ DEFAULT ]

<filespec> [ ,...n ]

}

Ниже представлены некоторые комментарии к синтаксису команды CREATE DATABASE.

**database\_name**

Имя новой базы данных. Имена баз данных должны быть уникальны внутри экземпляра SQL Server и должны соответствовать правилам для идентификаторов.

Аргумент database\_name может иметь максимальную длину 128 символов, если для файла журнала не указано логическое имя. Если логическое имя файла не указано, то SQL Server формирует для журнала имена logical\_file\_name и os\_file\_name путем добавления суффикса к database\_name. Это ограничивает длину аргумента database\_name 123 символами, чтобы формируемое логическое имя файла было не длиннее 128 символов.

Если имя файла данных не указано, то SQL Server использует аргумент database\_name в качестве имен logical\_file\_name и os\_file\_name. Путь по умолчанию берется из реестра. Его можно изменить в «Свойствах сервера» (страница «Параметры базы данных») в Management Studio. Изменение пути по умолчанию требует перезапуска SQL Server.

**ON**

Указывает, что дисковые файлы, используемые для хранения разделов данных в базе данных, определяются явно. Параметр ON необходимо применять, если за ним следует список элементов <filespec> с разделителями-запятыми, которые определяют файлы данных первичной файловой группы. За списком файлов в первичной файловой группе может следовать необязательный список элементов <filegroup> с разделителями-запятыми, которые определяют файловые группы пользователей и принадлежащие им файлы.

**PRIMARY**

Указывает, что связанный список <filespec> определяет первичный файл. Первый файл, указанный в элементе <filespec> в первичной файловой группе, становится первичным файлом. В базе данных может быть только один первичный файл.

Если параметр PRIMARY не указан, то первый файл списка в инструкции CREATE DATABASE становится первичным файлом.

**LOG ON**

Указывает, что дисковые файлы, используемые для хранения журнала базы данных, то есть файлы журналов, определяются явно. За параметром LOG ON следует список элементов <filespec> с разделителями-запятыми, которые определяют файлы журналов. Если параметр LOG ON не указан, автоматически создается один файл журнала, размер которого определяется большей из следующих двух величин: 512 КБ или 25 процентов от суммы размеров всех файлов с данными в базе данных. Этот файл помещается в местоположение для журнала по умолчанию.

**<filespec>**

Управляет свойствами файла.

**NAME logical\_file\_name**

Логическое имя, используемое в SQL Server при указании ссылки на файл. Аргумент **logical\_file\_name** должен быть уникальным в базе данных и должен соответствовать правилам для идентификаторов. Имя может быть символом или константой Юникода, а также обычным идентификатором или идентификатором с разделителями.

**FILENAME { 'os\_file\_name' }**

Задает физическое имя **-** путь и имя файла, используемые операционной системой при создании файла. Файл должен находиться на одном из следующих устройств: на локальном сервере, где установлен SQL Server, в сети хранения данных SAN или в сети на основе iSCSI. Указанный путь должен существовать до выполнения инструкции CREATE DATABASE.

Для файловой группы FILESTREAM параметр FILENAME указывает путь, где будут храниться данные FILESTREAM. Должен существовать путь вплоть до последнего каталога, но последний каталог существовать не должен. Например, если указать путь "D:\DATA\ FilestreamData ", папка " D:\DATA" должна существовать до запуска инструкции CREATE DATABASE, а папка "FilestreamData" - не должна.

Для файловой группы FILESTREAM может быть только один файл, причем файловую группу и файл необходимо создавать в одной инструкции.

Параметры **SIZE, MAXSIZE и FILEGROWTH** не применяется к файловой группе FILESTREAM и недоступны, если путь к файлу указан в формате UNC.

Если файл находится в необработанной секции (**raw partition**), аргумент os\_file\_name должен указывать только букву диска существующей необработанной секции. В каждой необработанной секции может быть создан только один файл.

Файлы данных не следует размещать в файловых системах со сжатием, за исключением случаев, когда файлы являются вторичными и доступны только для чтения или вся база данных доступна только для чтения. Файлы журналов ни в коем случае не должны размещаться в сжатых файловых системах.

**SIZE size**

Задает начальный размер файла. Параметр SIZE не может указываться, если аргумент os\_file\_name задан как путь в формате UNC.

Если аргумент size не задан для первичного файла, то компонент Database Engine использует размер первичного файла, указанный в базе данных model. Когда указан вторичный файл данных или журнала, но параметр size для файла не указан, компонент Database Engine задает размер файла равным 1 МБ. Размер, указанный для первичного файла, не должен быть менее размера первичного файла базы данных model.

Можно использовать суффиксы килобайт (KB), мегабайт (MB), гигабайт (GB) и терабайт (TB). По умолчанию — MБ. Аргумент Size имеет тип integer. Для значений, превышающих 2 147 483 647, используются более крупные единицы измерения.

**MAXSIZE max\_size**

Задает максимальный размер, до которого может расти файл. Параметр **MAXSIZE** нельзя указывать, если аргумент os\_file\_name задан как путь в формате UNC. Можно использовать суффиксы KB, MB, GB и TB. По умолчанию — MБ. Укажите целое число (без дробной части). Если аргумент max\_size не указан, размер файла будет увеличиваться до заполнения диска. Аргумент Max\_size имеет тип integer. Для значений, превышающих 2 147 483 647, используются более крупные единицы измерения. **UNLIMITED у**казывает, что файл может расти вплоть до заполнения диска. В SQL Server файл журнала, для которого задано неограниченное увеличение размера, имеет максимальный размер 2 ТБ, а файл данных — 16 ТБ.

**FILEGROWTH growth\_increment**

Задает автоматическое приращение размера файла. Значение параметра FILEGROWTH для файла не может превосходить значение параметра MAXSIZE. Параметр FILEGROWTH нельзя указывать, если аргумент os\_file\_name задан как путь в формате UNC.

Значение может быть указано в килобайтах, мегабайтах, гигабайтах, терабайтах или процентах (%). Если указано число без суффикса MB, KB или %, то по умолчанию используется MB. Если размер указан в процентах (%), то шаг роста это заданная часть в процентах от размера файла во время этого файла. Указанный размер округляется до ближайших 64 КБ.

Значение 0 указывает, что автоматическое приращение отключено и добавление пространства запрещено.

Если параметр FILEGROWTH не задан, значением по умолчанию является 1 МБ для файлов данных и 10% для файлов журналов, минимальное значение — 64 КБ.

**<filegroup>**

Управляет свойствами файловой группы. Файловая группа не может указываться для моментального снимка базы данных.

**FILEGROUP filegroup\_name**

Логическое имя файловой группы. Аргумент filegroup\_name должен быть уникальным в базе данных и не может быть именем PRIMARY или PRIMARY\_LOG, предоставленным системой. Имя может быть символом или константой Юникода, а также обычным идентификатором или идентификатором с разделителями. Имя должно соответствовать правилам для идентификаторов.

**CONTAINS FILESTREAM**

Указывает, что файловая группа хранит большие двоичные объекты (BLOB) FILESTREAM в файловой системе.

**DEFAULT**

Задаёт указанную файловую группу как файловую группу по умолчанию в базе данных.

***Пример 1***

В следующем примере создаётся база данных с именем TestDatabase. В ней создаются 3 файла – первичный файл данных (логическое имя "TestDatabaseData" в первичной файловой группе), вторичный файл данных (логическое имя "TestDatabaseUserFile" в файловой группе USERFG), FILESTREAM (логическое имя " Stream1" в файловой группе FileStreamG1) и файл журнала транзакций "TestDatabase\_log". Для каждого файла задаётся физическое имя файла (путь в файловой системе), начальный размер и автоматическое приращение.

CREATE DATABASE [TestDatabase] ON PRIMARY

( NAME = 'TestDatabaseData', FILENAME = N'D:\MSSQL\DATA\TestDatabaseData.mdf' , SIZE = 3072KB , FILEGROWTH = 1024KB ),

FILEGROUP [USERFG]

( NAME = 'TestDatabaseUserFile', FILENAME = N'D:\MSSQL\DATA\TestDatabaseUserFile.ndf' , SIZE = 30MB , FILEGROWTH = 2MB ),

FILEGROUP [FileStreamG1] CONTAINS FILESTREAM( NAME = Stream1, FILENAME = N'D:\MSSQL\DATA\FilestreamData')

LOG ON

( NAME = 'TestDatabase\_log', FILENAME = N'D:\MSSQL\DATA\TestDatabase\_log.ldf' , SIZE = 1024KB , FILEGROWTH = 10%)

Внимание! Чтобы настроить filestream, необходимо установить соответствующие свойства экземпляра сервера SQL Server в его свойствах (рис. 2.1), затем выполнить команду

EXEC sp\_configure filestream\_access\_level, 2

RECONFIGURE

и перезапустить сервис.

**2.2 Команда изменения настроек базы данных**

Изменение настроек базы данных производится с помощью команды **ALTER DATABASE**.

Данная команда позволяет добавлять или удалять файлы и файловые группы из базы данных, а также изменять параметры базы данных или ее файлов и файловых групп.

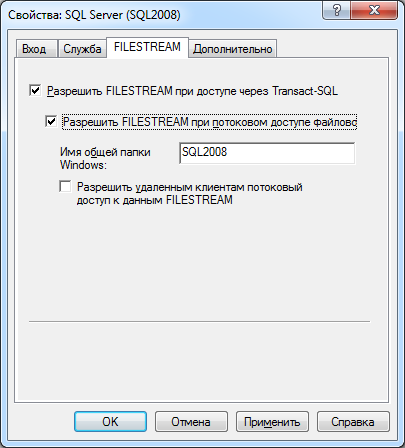


Рис. 2.1. Окно настройки filestream в окне свойств экземпляра SQL Server

ALTER DATABASE database\_name

{

   MODIFY NAME = new\_database\_name

  | COLLATE collation\_name

  | <add\_or\_modify\_files>

| <add\_or\_modify\_filegroups>

}

<add\_or\_modify\_files>::=

{

ADD FILE <filespec> [ ,...n ]

[ TO FILEGROUP { filegroup\_name } ]

| ADD LOG FILE <filespec> [ ,...n ]

| REMOVE FILE logical\_file\_name

| MODIFY FILE <filespec>

}

<filespec>::=

(

NAME = logical\_file\_name

[ , NEWNAME = new\_logical\_name ]

[ , FILENAME = {'os\_file\_name' | 'filestream\_path' } ]

[ , SIZE = size [ KB | MB | GB | TB ] ]

[ , MAXSIZE = { max\_size [ KB | MB | GB | TB ] | UNLIMITED } ]

[ , FILEGROWTH = growth\_increment [ KB | MB | GB | TB| % ] ]

)

<add\_or\_modify\_filegroups>::=

{

ADD FILEGROUP filegroup\_name [ CONTAINS FILESTREAM ]

| REMOVE FILEGROUP filegroup\_name

| MODIFY FILEGROUP filegroup\_name

{

<filegroup\_updatability\_option>

|DEFAULT

| NAME = new\_filegroup\_name

}

}

<filegroup\_updatability\_option>::=

{

{ READONLY | READWRITE }

| { READ\_ONLY | READ\_WRITE }

}

Ниже представлены некоторые комментарии к синтаксису команды ALTER DATABASE.

**database\_name** - имя изменяемой базы данных.

**MODIFY NAME = new\_database\_name** - присваивает базе данных новое имя, указанное в аргументе **new\_database\_name**.

**COLLATE collation\_name** - определяет параметры сортировки для базы данных. Аргумент collation\_name может быть либо именем параметров сортировки Windows, либо именем параметров сортировки SQL. Если аргумент не указан, базе данных будут назначены параметры сортировки экземпляра SQL Server.

**<** **add\_or\_modify\_files >::=**

**ADD FILE**

Добавляет файл к базе данных.

**TO FILEGROUP { filegroup\_name }**

Указывает файловую группу, к которой необходимо добавить указанный файл. Чтобы отобразить текущую файловую группу и узнать, какая файловая группа в данный момент установлена по умолчанию, используйте представление каталога **sys.filegroups**.

**ADD LOG FILE**

Добавляет файл журнала в указанную базу данных.

**REMOVE FILE logical\_file\_name**

Удаляет логическое описание файла из экземпляра SQL Server и физический файл. Файл не может быть удален, если он не пуст. **logical\_file\_name** **-** логическое имя, используемое в SQL Server при обращении к файлу.

**MODIFY FILE**

Указывает файл, который должен быть изменён. Одновременно может быть изменено только одно свойство файла. Предложение **NAME** всегда должно быть указано, чтобы определить, какой файл будет изменён. Если указано предложение **SIZE**, новый размер файла должен быть больше, чем текущий.

Чтобы изменить логическое имя файла данных или файла журнала, указывается логическое имя файла, который будет переименован, в предложении **NAME** и новое логическое имя для файла в предложении **NEWNAME**.

Например:

MODIFY FILE ( NAME = logical\_file\_name, NEWNAME = new\_logical\_name )

Чтобы переместить файл данных или файл журнала в новое расположение, указывается текущее логическое имя файла в предложении **NAME** и новый путь и имя файла в операционной системе в предложении **FILENAME**.

Например:

MODIFY FILE

( NAME = logical\_file\_name, FILENAME = ' new\_path/os\_file\_name ' )

**<add\_or\_modify\_filegroups>::=**

**ADD FILEGROUP filegroup\_name**

Добавляет файловую группу в базу данных.

**REMOVE FILEGROUP filegroup\_name**

Удаляет файловую группу из базы данных. Файловая группа не может быть удалена, пока она не пустая. Вначале удалите из файловой группы все файлы.

**MODIFY FILEGROUP filegroup\_name{ <filegroup\_updatability\_option> | DEFAULT | NAME = new\_filegroup\_name }**

Изменяет файловую группу, меняя ее состояние на READ\_ONLY или READ\_WRITE, делая ее файловой группой по умолчанию для базы данных или изменяя имя файловой группы.

**DEFAULT**

Изменяет файловую группу по умолчанию базы данных на аргумент **filegroup\_name**.

**NAME = new\_filegroup\_name**

Изменяет имя файловой группы на аргумент **new\_filegroup\_name**.

**<filegroup\_updatability\_option>** **-** устанавливает свойство «только для чтения» или «чтение и запись» для файловой группы.

* + **READ\_ONLY | READONLY**

Определяет, что файловая группа находится в состоянии только для чтения. Изменение ее объектов запрещено. Первичную файловую группу перевести в состояние только для чтения нельзя. Чтобы изменить это состояние, необходимо обладать монопольным доступом к базе данных.

* + **READ\_WRITE | READWRITE**

Определяет, что файловая группа находится в состоянии **READ\_WRITE**. Разрешено изменять объекты в файловой группе. Чтобы изменить это состояние, необходимо обладать монопольным доступом к базе данных. Ключевое слово READWRITE будет удалено в будущей версии Microsoft SQL Server. Вместо него следует используовать READ\_WRITE.

**<set\_database\_options >::=**

Одна из опций базы данных, приведенных выше, или их комбинация.

**2.3 Примеры изменения настроек базы данных**

***Пример 2:***

В следующем примере к базе данных TestDatabase добавляется файл данных размером 5 МБ.

ALTER DATABASE TestDatabase

ADD FILE

(

NAME = NewDataFile,

FILENAME = 'D:\MSSQL\DATA\NewDataFile.ndf',

SIZE = 5MB,

MAXSIZE = 100MB,

FILEGROWTH = 5MB

)

***Пример 3:***

В следующем примере в базе данных TestDatabase создается файловая группа NewUserFG и добавляется два файла по 5 МБ в эту файловую группу.

ALTER DATABASE TestDatabase

ADD FILEGROUP NewUserFG;

GO

ALTER DATABASE TestDatabase

ADD FILE

(

NAME = NewDataFile2,

FILENAME = 'D:\MSSQL\DATA\NewDataFile2.ndf',

SIZE = 5MB,

MAXSIZE = 100MB,

FILEGROWTH = 5MB

),

(

NAME = NewDataFile3,

FILENAME = 'D:\MSSQL\DATA\NewDataFile3.ndf',

SIZE = 5MB,

MAXSIZE = 100MB,

FILEGROWTH = 5MB

)

TO FILEGROUP newUserFG;

***Пример 4:***

В следующем примере к базе данных TestDatabase добавляется файл журнала NewLogFile размером 5 МБ.

ALTER DATABASE TestDatabase

ADD LOG FILE

(

NAME = NewLogFile,

FILENAME = 'D:\MSSQL\DATA\NewLogFile.ldf',

SIZE = 5MB,

MAXSIZE = 100MB,

FILEGROWTH = 5MB

);

***Пример 5:***

В следующем примере из базы данных TestDatabase удаляется один из ранее добавленных файлов с логическим именем NewDataFile3. Прежде чем файл может быть удалён, он должен быть совершенно пуст. Для переноса данных из одного файла данных в другие файлы той же файловой группы используется инструкция **DBCC SHRINKFILE** с указанием предложения **EMPTYFILE**.

-- перемещает данные из файла NewDataFile3

-- в другие файлы файловой группы newUserFG

DBCC SHRINKFILE (NewDataFile3, EMPTYFILE);

GO

-- удаляет пустой файл

ALTER DATABASE TestDatabase

REMOVE FILE NewDataFile3;

**3 Удаление базы данных**

Удаление базы данных осуществляется командой **DROP DATABASE**.

DROP DATABASE database\_name

При удалении базы данных она удаляется из экземпляра SQL Server, а ее физические файлы удаляются с диска. Если база данных или один из ее файлов во время удаления находится в автономном режиме, файлы с диска не удаляются, эти файлы можно удалить вручную при помощи обозревателя Windows.

Удалить базу данных, которая используется в текущий момент времени, невозможно. Для отключения пользователей от базы данных можно использовать инструкцию ALTER DATABASE для перевода базы данных в режим SINGLE\_USER.

База данных может быть удалена независимо от ее состояния: в автономном режиме, доступна только для чтения, подозрительная и т. д. Для просмотра текущего состояния базы данных можно воспользоваться представлением каталога **sys.databases**.

Удалённая база данных может быть повторно создана только с помощью восстановления из резервной копии.

***Пример 6:***

В следующем примере удаляется ранее созданная база данных TestDatabase.

DROP DATABASE TestDatabase ;

**4. Создание таблиц**

**4.1 Команда CREATE TABLE**

**CRЕАТЕ TABLE** – одна из самых объёмных команд Transact-SQL. В общем виде она определяется следующим образом:

CRЕАТЕ TABLE

[ database\_name.[owner].|owner.] table\_name

( { <column\_definition>

| column\_name AS computed\_colurm\_expression

| <table\_constraint > ::= [CONSTRAINT constraint\_name ]}

| [ { PRIMARY KEY | UNIQUE }[ ,... n ])

[ ON {filegroup | DEFAULT}]

{ TEXTIMAGE\_ON {filegroup | DEFAULT } ]

В этой конструкции:

* database\_name – имя базы данных, где создается данная таблица;
* owner – владелец таблицы. По умолчанию владельцем таблицы является тот, кто ее создает. Указывать другого владельца можно лишь в том случае, если вы являетесь членом группы (роли) sysadmin;
* table\_name – имя создаваемой таблицы. Длина имени таблицы не может превышать 128 символов. Если имя таблицы начинается со знаков "#" или "##", то будут созданы соответственно локальная или глобальная временные таблицы. Полная длина имени в этом случае не может превышать 116 символов;
* column\_name – имя столбца таблицы;
* computed\_colurm\_expression – выражения для вычисляемых столбцов, которые могут присутствовать в списках столбцов оператора select. Они используются в индексах и при формировании первичного ключа. Однако вычисляемые столбцы не могут быть внешними ключами и присутствовать в операторах update и insert;
* ON {filegroup | DEFAULT} – ключевые слова для указания группы файлов**,** где будет храниться таблица. Если данный параметр опущен или стоит on default, то создаваемая таблица будет размещена в группе файлов умолчанию;
* TEXTIMAGE\_ON {filegroup | DEFAULT } – ключевые слова, с помощью которых можно определить группу файлов для хранения столбцов с типами данных text, ntext, image (размещаться в отдельной группе файлов);
* <column\_definition> – конструкция для определения каждого столбца. Ее формат имеет следующий вид:

<column\_definition>::= { column\_name data\_type

[COLLATE <collation\_name>]

[ [ DEFAULT constant\_expression]

| [ IDENTITY [(seed, increment)

[ NOT FOR REPLICATION ]]]] [ ROWGUIDCOL ]

[ <column\_constraint> ] [ ... n ]

Здесь:

* data\_type – тип данных столбца. Может быть указан как стандартный, так и пользовательский тип данных. При задании пользовательского типа данных следует иметь в виду, что возможность хранения значения null может быть определена как на уровне столбца, так и на уровне данных;
* COLLATE <collation\_name> – конструкция, где создается сопоставление (имя) столбца. Данный параметр имеет смысл только для столбцов типа char, varcahar, text, nchar, nvarchar, ntext;
* DEFAULT constant\_expression – конструкция, определяющая значение по умолчанию для данного столбца. Это может быть константа или встроенная функция, например, newid();
* IDENTITY – ключ, определяющий, что данный столбец будет столбцом-счетчиком, при этом seed – его начальное значение, a increment – шаг приращения;
* NOT FOR REPLICATION – опция, предписывающая серверу не выполнять действия, если они производятся подсистемой репликаций;
* ROWGUIDCOL – опция, устанавливающая, что данный столбец будет хранить глобальный идентификатор. Для такого столбца автоматически определяется значение по умолчанию функцией newid();
* column\_constraint – конструкция ограничений значений столбца, определяемая следующим образом:

< column\_\_constraint > :: =

[ CONSTRAINT constraint\_name ] { [ NULL | NOT NULL ]

| [ { PRIMARY KEY | UNIQUE }

[ CLUSTERED | NONCLUSTERED ]

[ WITH FILLFACTOR = fillfactor ]

[ ON {filegroup | DEFAULT} ] ] ]

| [ [ FOREIGN KEY ]

REFERENCES ref\_table [ ( ref\_column ) ]

[ ON DELETE { CASCADE ] NO ACTION } ]

[ ON UPDATE { CASCADE | NO ACTION } ]

[ NOT FOR REPLICATION ] ]

CHECK [ NOT FOR REPLICATION ]

(logical\_\_expression) }

В этой конструкции:

* CONSTRAINT constraint\_name – название ограничения на столбец;
* NULL | NOT NULL –— опция, определяющая возможность хранения в столбце значение неопределенных значений null;
* PRIMARY KEY – опция, указывающая, что столбец является первичным ключом. Если предполагается, что первичный ключ будет состоять из нескольких полей, то его следует определять на уровне таблицы (см. пункт table\_constraint);
* UNIQUE – опция, гарантирующая, уникальность каждого значения в данном столбце. При этом будет автоматически создан индекс по данному полю;
* CLUSTERED | NONCLUSTERED – указатель на необходимость создания кластерного или некластерного индекса. По умолчанию для первичного ключа создается кластерный индекс, а для столбца со свойством unique – некластерный;
* WITH FILLFACTOR = fillfactor – конструкция, где задается фактор заполнения поля в процентах;
* ON {filegroup | DEFAULT}– конструкция, определяющая группу файлов для хранения создаваемого для данного столбца индексного файла;
* FOREIGN KEY – опция, устанавливающая данных столбец внешним ключом. Последующее выражение REFERENCES ref\_table [(ref\_column) определяет ссылку на таблицу (и ее столбцы) внешним ключом, для которой является создаваемый столбец. Ссылку можно установить только на столбец, определенный как первичный ключ или ключ unique;
* ON DELETE {CASCADE | NO ACTION} – конструкция, определяющая сценарий удаления записей из родительской таблицы (при условии что данный столбец определен как внешний ключ). Если задать опцию cascade, то строка будет удалена при удалении соответствующей записи из родительской таблицы. Если установлена опции no action, то в этом случае сервер выдаст сообщение об ошибке и произведет откат транзакции;
* ON UPDATE {CASCADE | NO ACTION} – опция для определения аналогичного предыдущему сценария, но только для команды update;
* CHECK– ключевое слово, после которого можно установить ограничение целостности по логическому выражению logical\_expression;
* tabie\_constraint – конструкция ограничений следующего вида:

< table\_constraint > ::= [ CONSTRAINT constraint\_name ]

{ [ { PRIMARY KEY | UNIQUE }

[ CLUSTERED | NONCLUSTERED ]

{ { column [ ASC | DESC ] [ ,... n ] ) }

[ WITH FILLFACTOR = fillfactor ]

[ ON { filegroup | DEFAULT } ] ]

FOREIGN KEY [ ( column [ ,... n ] ) ]

REFERENCES ref\_table [ ( ref\_column [ , ... n ])]

[ ON DELETE { CASCADE | NO ACTION )]

[ ON UPDATE { CASCADE | NO ACTION }]

[ NOT FOR REPLICATION]

CHECK [ NOT FOR REPLICATION ] (search\_conditions)}

Эта конструкция почти полностью совпадает с предыдущей (column\_constraint), поэтому отметим лишь отсутствующие опции:

* column [ASC | DESC] [ ,... n ] ) – список столбцов, на которые необходимо наложить ограничения. Далее указывается тип сортировки, производимой при индексации по данному столбцу (ASC или DESC).

**4.2 Изменение структуры таблицы**

Изменить структуру таблицы можно с помощью команды alter table. Ниже представлен ее полный формат.

ALTER TABLE table

([ ALTER COLUMN column\_name

new\_data\_type [(precision [, scale])]

[ COLLATE <collation\_name> ]

[ NULL | NOT NULL ] | { ADD | DROP } ROWGUIDCOL } ]

| ADD

{ [ < column\_definition > ]

| column\_name AS computed\_coluinn\_expression) [, ... n]

| [ WITH CHECK | WITH NOCHECK ] ADD

{ < table\_constraint > } [ ,... n ]

| DROP { [ CONSTRAINT ] constraint\_name | COLUMN column }

[ , ... n ]

| { CHECK | NOCHECK } CONSTRAINT

{ ALL | constraint\_name [ ,...n ] }

| { ENABLE | DISABLE } TRIGGER

{ ALL | trigger\_name [ ,... n ] }}

<column\_\_definition > :: =

{ column\_name data\_type }

[ [ DEFAULT constant\_expression ] [ WITH VALUES ]

| [ IDENTITY [(seed , increment) [NOT FOR REPLICATION ]]]]

[ ROWGUIDCOL ]

[ COLLATE < collation\_name > ]

[ < column\_constraint > ] [ ,... n ]

< column\_constraint > :: =

[ CONSTRAINT constraint\_name ]

{ [ NULL | NOT NULL ]

[ { PRIMARY KEY [ UNIQUE }

[ CLUSTERED | NONCLUSTERED ]

[ WITH FILLFACTOR = fillfactor ]

[ ON { filegroup | DEFAULT } ]]

| [ [ FOREIGN KEY ]

REFERENCES ref\_table [ ( ref\_column ) ]

[ ON DELETE { CASCADE | NO ACTION } ]

[ ON UPDATE { CASCADE | NO ACTION } ]

[ NOT FOR REPLICATION ]]

| CHECK [ NOT FOR REPLICATION ]

( logical\_expression ) }

< table\_constraint > :: =

[ CONSTRAINT constraint\_name ]

{ [ { PRIMARY KEY | UNIQUE }

[CLUSTERED | NONCLUSTERED ]

{ ( column [ ,... n ] ) }

[ WITH FILLFACTOR = fillfactor ]

[ ON {filegroup | DEFAULT } ]]

| FOREIGN KEY

[ ( column [ ,... n ] ) ]

REFERENCES ref\_table [ ( ref\_column [ ,... n ] ) ]

[ ON DELETE { CASCADE | NO ACTION } ]

[ ON UPDATE { CASCADE | NO ACTION } ]

[ NOT FOR REPLICATION ]

| DEFAULT constant\_expression

[ FOR column ] [ WITH VALUES ]

| CHECK [ NOT FOR REPLICATION ]

( search\_conditions ) }

Нетрудно заметить, что ее формат в значительной степени совпадает с форматом команды

CREATE TABLE, в силу чего сделаем лишь минимальные пояснения:

* table – имя или полное имя таблицы, структура которой будет изменяться;
* ALTER COLUMN column\_name – имя модифицируемого столбца. Модификация столбца невозможна, если:
* столбец имеет тип TEXT, NTEXT, IMAGE, TIMESTAMP;
* у столбца установлено свойство ROWGUIDCOL;
* столбец является вычисляемым или используется при вычислении другого столбца;
* столбец участвует в репликации;
* столбец включен в индекс. В этом случае придется сначала удалить индекс, и лишь затем изменять столбец;
* столбец необходим для сбора статистических данных. Здесь следует сначала выполнить команду DROP STATISTICS;
* столбец участвует в ограничениях. Как и в предыдущих случаях ограничения предварительно следует удалить, и только затем корректировать столбец;
* столбцу предписано значение по умолчанию;
* new\_data\_type – новый тип данных модифицируемого столбца. Следует учитывать следующее моменты:
* при изменении типа данных существующие значения неявно конвертируются в новый тип данных;
* новый тип данных не может быть типом TIMESTAMP;
* по умолчанию столбцу будет предписана возможность принимать значение null;
* если изменяемый столбец поддерживает свойство ident, то результирующий столбец также будет поддерживать это свойство;
* игнорируется установка SET ARITHABORT;
* COLLATE <collation\_name> – конструкция, позволяющая изменить сопоставление. Модифицировать сопоставление можно только для столбцов типа CHAR, NCHAR, VARCHAR, TEXT, NTEXT, NVARCHAR. Изменение сопоставления невозможно в следующих случаях:
* на столбец наложено ограничение СНEАСК ИЛИ FOREIGN KEY или на него ссылается вычисляемый столбец;
* столбец индексирован или используется для статистики;
* на столбец ссылаются функции или представления;
* {ADD | DROP} ROWGUIDCOL – опция, определяющая действие со свойством rowguidcol (добавить или удалить);
* ADD – ключевое слово, означающее, что следует добавить новый столбец (в том числе и вычисляемый) или ограничение;
* with check | with nocheck – конструкции, предписывающие серверу при вставке новой строки проверять или не проверять новое значение на соответствие ограничениям (check или foreign key);
* DROP {[CONSTRAINT] constraint\_name | COLUMN column} – конструкции, указывающие серверу на необходимость удаления ограничения или столбца таблицы;
* { check | nocheck } constraint — опция, определяющая необходи­мость задействовать или блокировать параметр constraint\_name;
* ALL — опция, устанавливающая, что параметры nocheck и check действуют на все ограничения целостности;
* { ENABLE | DISABLE } TRIGGER — конструкция, определяющая состояние триггера с именем trigger\_name (задействовать/блокировать);
* ALL – опция, указывающая задействовать/заблокировать все триггеры данной таблицы;
* WITH VALUES – опция, предписывающая серверу заменять в столбце значения NULL на значения по умолчанию;
* For column – ключ для создания столбца, в котором должно быть определено значение по умолчанию.

**5 Задание для самостоятельной работы**

***Задание 1.***

Изменить структуры таблиц базы данных DBLab согласно требованиям, представленным в табл. 5.1. Изменение структуры всех таблиц производить с помощью запросов на языке Transact-SQL. Результаты представить в виде файла TablesAlter.sql, который при запуске позволит автоматически выполнить указанные действия.

Таблица 5.1

Спецификация добавляемых полей в таблицы базы данных

|  |  |  |
| --- | --- | --- |
| **Название добавляемого поля** | **Смысл** | **Примечание (формула для расчетного поля)** |
| **Таблица для добавления полей – «ex\_Product»** | | |
| PercentProduct | Процент выполненного производства по передельному чугуну | Вычислить по формуле:  (convert(decimal(5,2),round(case when ([PlanCalendar] > 0) then ([Sliv] / [PlanCalendar] \* 100.0) end,2))) |
| Shlak | Количество шлака, т | Вычислить по формуле:  (convert(decimal(12,3),([SlivDirty] - [Sliv]))) |
| **Таблица для добавления полей – «ex\_StandDetail»** | | |
| period\_sec | Период простоя в течении месяца в секундах | Вычислить по формуле:  (convert(int,datediff(second,[start],[finish]))) |
| TotalMin | Общая продолжительность простоя в минутах | Вычислить по формуле:  (convert(int,round((datediff(second,[start],[finish]) / 60.0),0))) |
| TotalDurationHour | Общая продолжительность простоя в часах | Вычислить по формуле:  (convert(decimal(5,2),round((convert(int,round((datediff(second,[start],[finish]) / 60.0),0)) / 60.0),2))) |
| TotalDurationSut | Общая продолжительность простоя в сутках | Вычислить по формуле:  (convert(decimal(5,2),round((convert(int,round((datediff(second,[start],[finish]) / 60.0),0)) / 1440.0),2))) |
| TotalHourMin | Общий простой, часы.минуты | Вычислить по формуле:  (convert(decimal(5,2),(floor((convert(int,round((datediff(second,[start],[finish]) / 60.0),0)) / 60)) + floor((convert(int,round((datediff(second,[start],[finish]) / 60.0),0)) - convert(int,round((datediff(second,[start],[finish]) / 60.0),0)) / 60 \* 60)) / 100.0))) |
| **Таблица для добавления полей – «ex\_ShedulerVipusk»** | | |
| loseVipusk | Процент потери выпусков | Вычислить по формуле:  (convert(decimal(6,2),round(case when ([cntPlanVipusk] > 0) then (100.0 \* [cntFactVipusk] / [cntPlanVipusk]) end,2))) |
| **Таблица для добавления полей – «ex\_ChemChugun»** | | |
| FeCh | Содержание FeCh в чугуне | Вычислить по формуле:  (convert(decimal(6,3),round((100.0 - ([Si] + [Mn] + [S] + [Ti] + [P] + [CR] + [C])),3))) |
| **Таблица для добавления полей – «ex\_ChemShlak»** | | |
| Osn1 | Основность шлака:  CaO/SiO2 | Вычислить по формуле:  (convert(decimal(6,3),case when ([SiO2] > 0) then ([CaO] / [SiO2]) end)) |
| Osn2 | Основность шлака:  (CaO+MgO)/SiO2 | Вычислить по формуле:  (convert(decimal(6,3),case when ([SiO2] > 0) then (([CaO] + [MgO]) / [SiO2]) end)) |
| Osn3 | Основность шлака:  (CaO+MgO)/(SiO2+Al2O3) | Вычислить по формуле:  (convert(decimal(6,3),case when ([SiO2] + [Al2O3] > 0) then (([CaO] + [MgO]) / ([SiO2] + [Al2O3])) end)) |
| **Таблица для добавления полей – «ex\_Koks»** | | |
| CMKi | Расход сухого металлургического кокса, т | Вычислить по формуле:  (convert(decimal(12,1),([CCKi] + [CKMi]))) |
| **Таблица для добавления полей – «ex\_ChemRuda»** | | |
| Osn1 | Основность материала CaO/SiO2 | Вычислить по формуле:  (convert(decimal(6,3),case when ([SiO2] > 0) then ([CaO] / [SiO2]) end)) |

Скрипт TablesAlter.sql со своими комментариями вставить в отчет по проекту.

***Задание 2.***

Средствами MS SQL Server 2008 автоматически получить скрипт на языке Transact-SQL для создания всех таблиц базы данных DBLab. Имя результирующего файла TablesCreate.sql.

Для подготовки скрипта с помощью MS SQL Server 2008 следует загрузить надстройку SQL Server Management Studio, подключиться к серверу, и для базы данных DBLab с помощью мыши вызвать контекстное меню «Tasks / Generate SQL Scripts…». В результате на экране будут последовательно появляться окна мастера создания скриптов:

* окно выбора типа объектов для генерации скрипта (рис. 5.1);
* окно выбора перечня таблиц (рис. 5.2);
* окно выбора местоположения и названия файла скрипта (рис. 5.3).

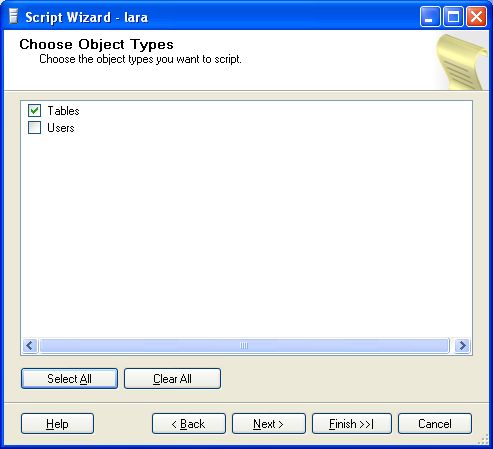


Рис. 5.1. Диалоговое окно выбора типа объектов для создания скриптов

Отметить тип объектов для генерации скрипта

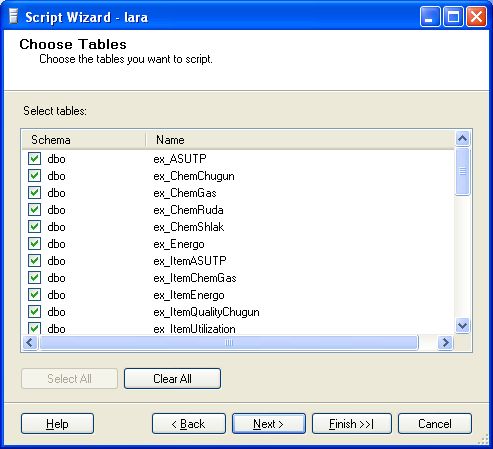
******

Рис. 5.2. Диалоговое окно выбора таблиц

Отметить выбор всех таблиц проекта

Указать местоположение и название файла скрипта

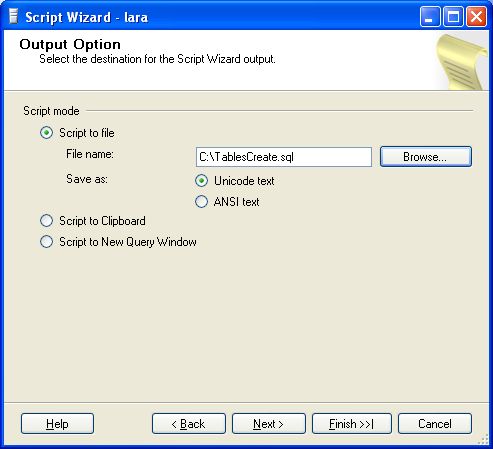
******

Рис. 5.3. Диалоговое окно выбора названия файла скрипта

Скрипт TablesCreate.sql со своими комментариями вставить в отчет по проекту, который подготовить в виде текстового документа MS Office Word.

ВЫПОЛНЕНИЕ ПРОЕКТА №2

**1. Создание хранимых процедур**

Хранимые процедуры – это процессы, выполнение которых происходит непосредственно на сервере баз данных.

Все хранимые процедуры в базе данных находятся в специально отведенном списке *Stored Procedures* раздела базы данных *Programmability*. Следует обратить внимание на перечень системных процедур, используемых при работе SQL-сервера, список которых находятся в группе System *Stored Procedures*.

Для создания новой процедуры выберите команду *New Stored Procedures* контекстногоменю*,* после чего на экране отобразится вкладка, в которой будет расположен текст шаблона для создания текста процедуры.

Рассмотрим использование хранимых процедур на примерах. Работать с пользовательской базой данных DBLab. Содержимое базы данных DBLab можно восстановить из архивной копии.

**Пример 1.** Создать хранимую процедуру, которая позволяет прочитать все данные из таблицы ex\_Product пользовательской базы данных DBLab.

В ходе выполнения хранимой процедуры идет обмен запросами между объектами базы данных – таблицами, хранимыми процедурами и пр. Порядок выполнения запросов, которые объекты посылают друг другу, лучше всего отображать на диаграмме взаимодействия. На диаграмме взаимодействия время откладывается сверху вниз. Сплошная вертикальная линия обозначает время жизни объекта. Если объект еще не создан к начальному моменту времени, представленному на диаграмме, то его вертикальная линия идет пунктиром вплоть до момента создания. Вертикальный прямоугольник говорит о том, что объект активен, т.е. обрабатывает некоторый запрос. Операция может посылать запросы другим объектам, они изображаются горизонтальной линией, указывающей на объект-получатель. Имя запроса показывается над стрелкой. Запрос на создание объекта представлен пунктирной стрелкой. Запрос отправителя самому себе изображается стрелкой, указывающей на сам этот объект.

На рис. 1.1 видно, что запрос от клиента инициирует запуск на SQL-сервере хранимой процедуры dbo.ex\_SEL\_ProductAll, которая в свою очередь обращается к таблице dbo.ex\_Product. После того, как таблица dbo.ex\_Product возвратит необходимые данные, хранимая процедура сформирует и возвратит клиенту таблицу с перечисленными столбцами.



Рис. 1.1 – Диаграмма взаимодействия выполнения  
хранимой процедуры dbo.ex\_SEL\_ProductAll

Хранимые процедуры лучше всего создавать и отлаживать в окне запросов, которое вызывается из контекстного меню «New Query…» соответствующей базы данных. Листинг хранимой процедуры **dbo.ex\_SEL\_ProductAll** приведен ниже.

SET QUOTED\_IDENTIFIER ON

GO

SET ANSI\_NULLS OFF

GO

/\*-----------------------------------------------------------

Проект "DBLab"

Пример обработки базы данных MS SQL Server 2008 R2

Разработчик: Лавров Владислав Васильевич

Функции:

Таблица "Производство передельного чугуна"

Чтение всех данных из таблицы

Пример вызова:

EXEC dbo.ex\_SEL\_ProductAll

\*/-----------------------------------------------------------

CREATE PROC dbo.ex\_SEL\_ProductAll

AS

SET NOCOUNT ON -- не передавать сообщения о количестве обработанных записей

SELECT F.dtFirstDay,

F.NPech,

F.PlanCalendar,

F.PlanStand,

F.Naliv,

F.Sliv,

F.PercentProduct,

F.NalivDirty,

F.SlivDirty,

F.Shlak

FROM dbo.ex\_Product AS F

GO

SET QUOTED\_IDENTIFIER OFF

GO

SET ANSI\_NULLS ON

GO

Представленный листинг начинается с установки параметров окружения, которые позволяют изменять поведение хранимых процедур. Настройка QUOTED\_IDENTIFIER определяет, интерпретировать ли строки в двойных кавычках как идентификаторы объектов (например, таблиц или столбцов). Установка ANSI\_NULLS определяет, разрешены ли не-ANSI сравнения с неопределенными (NULL) значениями, что является особенно важным для хранимых процедур, параметры которых могут иметь значения NULL.

Комментарии к хранимой процедуре должны содержать пример вызова процедуры для проверки ее работоспособности. С помощью SQL-команды EXEC осуществляется запуск хранимых процедур, причем весь процесс их выполнения происходит на самом компьютере-сервере. Использование хранимых процедур существенно отражается на производительности компьютеров рабочих станций. Другими словами, данная технология позволяет использовать компьютеры рабочих станций на уровне клавиатурного ввода, а все основные задачи при этом перекладываются на компьютер-сервер.

Следующий пример использования хранимых процедур будет использовать в процессе работы передаваемые параметры.

**Пример 2.** Создать хранимую процедуру, которая позволяет прочитать данные из таблицы ex\_Product пользовательской базы данных DBLab за произвольный месяц.

На рис. 1.2 показана диаграмма взаимодействия объектов при работе хранимой процедуры dbo.ex\_SEL\_Product. Особенностью является использование параметра @dtFirstDay, который позволяет вернуть клиенту отфильтрованные по дате данные.



Рис. 1.2 – Диаграмма взаимодействия выполнения  
хранимой процедуры dbo.ex\_SEL\_Product

Листинг хранимой процедуры **dbo.ex\_SEL\_Product** приведен ниже.

SET QUOTED\_IDENTIFIER ON

GO

SET ANSI\_NULLS OFF

GO

/\*-----------------------------------------------------------

Проект "DBLab"

Пример обработки базы данных MS SQL Server 2008 R2

Разработчик: Лавров Владислав Васильевич

Функции:

Таблица "Производство передельного чугуна"

Чтение данных из таблицы

Входные данные:

@dtFirstDay - первый день отчетной даты

Пример вызова:

EXEC dbo.ex\_SEL\_Product @dtFirstDay='2007.01.01'

\*/-----------------------------------------------------------

CREATE PROC dbo.ex\_SEL\_Product

@dtFirstDay smalldatetime

AS

SET NOCOUNT ON -- не передавать сообщения о количестве обработанных записей

SELECT F.dtFirstDay,

F.NPech,

F.PlanCalendar,

F.PlanStand,

F.Naliv,

F.Sliv,

F.PercentProduct,

F.NalivDirty,

F.SlivDirty,

F.Shlak

FROM dbo.ex\_Product AS F

WHERE F.dtFirstDay = @dtFirstDay

GO

SET QUOTED\_IDENTIFIER OFF

GO

SET ANSI\_NULLS ON

GO

Здесь следует обратить внимание на использование переменных-параметров, принимаемых процедурой. Так, каждая переменная внутри хранимой процедуры описывается следующим образом

@имя\_переменной тип\_данных

Если в процедуру передастся несколько параметров, то они указываются через запятую после ее имени. В нашем случае в процедуру передается переменная @dtFirstDay.

Представленные выше процедуры не полностью отражают средства использования хранимых процедур. Максимально достоинства данной технологии можно оценить, используя операторы управления.

Следующая конструкция позволяет определить выполнение операций в зависимости от определенного условия:

IF выражение

операторы

[ELSE]

[IF выражение]

операторы

**Пример 3.** Написать хранимую процедуру, которая вставляет строку в таблицу ex\_Product.

Диаграмма взаимодействия объектов при выполнении хранимой процедуры представлена на рис. 1.3.



Рис. 1.3 – Диаграмма взаимодействия выполнения  
хранимой процедуры dbo.ex\_INS\_Product

Листинг хранимой процедуры **dbo.ex\_INS\_Product** приведен ниже.

SET QUOTED\_IDENTIFIER ON

GO

SET ANSI\_NULLS OFF

GO

/\*-----------------------------------------------------------

Проект "DBLab"

Пример обработки базы данных MS SQL Server 2008 R2

Разработчик: Лавров Владислав Васильевич

Функции:

Таблица "Производство передельного чугуна"

Вставка строки в таблицу

Входные данные:

@dtFirstDay - первый день отчетной даты

@NPech - номер печи

@PlanCalendar - план календарный по передельному чугуну, т

@PlanStand - План по передельному чугуну c учетом простоев, т

@Naliv - налив передельного чистого чугуна, т

@Sliv - слив передельного чистого чугуна, т

@NalivDirty - налив передельного грязного чугуна (со шлаком), т

@SlivDirty - слив передельного грязного чугуна (со шлаком), т

Пример вызова:

EXEC dbo.ex\_INS\_Product @dtFirstDay='2007.01.01', @NPech = 1, @PlanCalendar = 99190, @PlanStand = 99000, @Naliv = 8000, @Sliv = 99247, @NalivDirty = 100000, @SlivDirty = 99800

-- Проверка вставленной строки

SELECT \* FROM dbo.ex\_Product WHERE dtFirstDay='2007.01.01' AND NPech = 1

\*/-----------------------------------------------------------

CREATE PROC dbo.ex\_INS\_Product

@dtFirstDay smalldatetime,

@NPech tinyint,

@PlanCalendar decimal(12, 3),

@PlanStand decimal(12, 3),

@Naliv decimal(12, 3),

@Sliv decimal(12, 3),

@NalivDirty decimal(12, 3),

@SlivDirty decimal(12, 3)

as

SET NOCOUNT OFF -- передавать сообщения о количестве обработанных записей

-- Проверить существование строки: если не существует, то вставить

IF NOT EXISTS ( SELECT 1 FROM dbo.ex\_Product WHERE dtFirstDay = @dtFirstDay AND NPech = @NPech )

BEGIN

INSERT INTO dbo.ex\_Product (

dtFirstDay,

NPech,

PlanCalendar,

PlanStand,

Naliv,

Sliv,

NalivDirty,

SlivDirty

)

VALUES (

@dtFirstDay,

@NPech,

@PlanCalendar,

@PlanStand,

@Naliv,

@Sliv,

@NalivDirty,

@SlivDirty

)

SET NOCOUNT ON -- отключить сообщения о количестве обработанных записей

-- Прочитать новую запись

SELECT F.dtFirstDay,

F.NPech,

F.PlanCalendar,

F.PlanStand,

F.Naliv,

F.Sliv,

F.PercentProduct,

F.NalivDirty,

F.SlivDirty,

F.Shlak

FROM dbo.ex\_Product AS F

WHERE F.dtFirstDay = @dtFirstDay AND F.NPech = @NPech

END

ELSE

SET NOCOUNT ON

GO

SET QUOTED\_IDENTIFIER OFF

GO

SET ANSI\_NULLS ON

GO

Следует обратить внимание на использование ключевого слова EXISTS, благодаря чему проверяется наличие строки данных, обусловленных предикатом.

Если имеется необходимость в использования нескольких подряд идущих операторов, то в этом случае необходимо воспользоваться следующей конструкцией:

BEGIN

операторы

END

Конструкция оператора CASE … END может заменить множество условных операторов, при этом она ведет себя как функция, т.е. ее можно вставлять в выражения в качестве переменной.

Далее продемонстрируем процедуры с использованием временных таблиц и некоторых функций преобразования.

**Пример 4.** Написать хранимую процедуру, которая производит расчет производства передельного чугуна с начала года до указанной даты.

Диаграмма взаимодействия объектов при выполнении хранимой процедуры представлена на рис. 1.4. Особенностью представленной процедуры является то, что при реализации использована временная таблица #Report, структура которой не меняется.



Рис. 1.4 – Диаграмма взаимодействия выполнения  
хранимой процедуры dbo.ex\_CALC\_Product\_1

Листинг хранимой процедуры **dbo.ex\_CALC\_Product\_1** приведен ниже.

SET QUOTED\_IDENTIFIER ON

GO

SET ANSI\_NULLS OFF

GO

/\*-----------------------------------------------------------

Проект "DBLab"

Пример обработки базы данных MS SQL Server 2008 R2

Разработчик: Лавров Владислав Васильевич

Функции:

Таблица "Производство передельного чугуна"

Расчет производства передельного чугуна с начала года до указанной даты.

При реализации использована временная таблица, структура которой не меняется

Входные данные:

@dtFirstDay - первый день отчетного месяца

Пример вызова:

EXEC dbo.ex\_CALC\_Product\_1 @dtFirstDay='2007.12.01'

\*/-----------------------------------------------------------

CREATE PROC dbo.ex\_CALC\_Product\_1

@dtFirstDay smalldatetime

AS

SET NOCOUNT ON -- отключить сообщения о количестве обработанных записей

SET ANSI\_WARNINGS OFF -- отключить вывод предупреждений

-- Декларировать временную переменную и присвоить дату c начала года

DECLARE @DateBegYear smalldatetime

SET @DateBegYear = Convert(smalldatetime, Convert(varchar, DatePart(year, @dtFirstDay)) + '.01.01')

-- Объявить временную таблицу

CREATE TABLE #Report (

dtFirstDay smalldatetime NOT NULL,

Descr char(40) NOT NULL,

PlanDP1 real NULL,

SlivDP1 real NULL,

PlanDP2 real NULL,

SlivDP2 real NULL,

PlanDP3 real NULL,

SlivDP3 real NULL,

PlanDP4 real NULL,

SlivDP4 real NULL,

PlanDP5 real NULL,

SlivDP5 real NULL,

PlanDP6 real NULL,

SlivDP6 real NULL,

PlanDP7 real NULL,

SlivDP7 real NULL,

PlanDP8 real NULL,

SlivDP8 real NULL,

PlanDP9 real NULL,

SlivDP9 real NULL,

PlanDP10 real NULL,

SlivDP10 real NULL,

PlanDP255 real NULL,

SlivDP255 real NULL

)

-- Заполнить временную таблицу данными по выплавке передельного чугуна

-- Каждую колонку заполнить запросом к таблице dbo.ex\_Product

INSERT INTO #Report (

dtFirstDay,

Descr,

PlanDP1, SlivDP1,

PlanDP2, SlivDP2,

PlanDP3, SlivDP3,

PlanDP4, SlivDP4,

PlanDP5, SlivDP5,

PlanDP6, SlivDP6,

PlanDP7, SlivDP7,

PlanDP8, SlivDP8,

PlanDP9, SlivDP9,

PlanDP10, SlivDP10,

PlanDP255, SlivDP255

)

SELECT DISTINCT A.dtFirstDay, 'Передельный',

PlanDP1 = (SELECT X.PlanCalendar FROM dbo.ex\_Product as X (nolock) WHERE X.dtFirstDay = A.dtFirstDay AND X.NPech = 1),

SlivDP1 = (SELECT X.Sliv FROM dbo.ex\_Product as X (nolock) WHERE X.dtFirstDay = A.dtFirstDay AND X.NPech = 1),

PlanDP2 = (SELECT X.PlanCalendar FROM dbo.ex\_Product as X (nolock) WHERE X.dtFirstDay = A.dtFirstDay AND X.NPech = 2),

SlivDP2 = (SELECT X.Sliv FROM dbo.ex\_Product as X (nolock) WHERE X.dtFirstDay = A.dtFirstDay AND X.NPech = 2),

PlanDP3 = (SELECT X.PlanCalendar FROM dbo.ex\_Product as X (nolock) WHERE X.dtFirstDay = A.dtFirstDay AND X.NPech = 3),

SlivDP3 = (SELECT X.Sliv FROM dbo.ex\_Product as X (nolock) WHERE X.dtFirstDay = A.dtFirstDay AND X.NPech = 3),

PlanDP4 = (SELECT X.PlanCalendar FROM dbo.ex\_Product as X (nolock) WHERE X.dtFirstDay = A.dtFirstDay AND X.NPech = 4),

SlivDP4 = (SELECT X.Sliv FROM dbo.ex\_Product as X (nolock) WHERE X.dtFirstDay = A.dtFirstDay AND X.NPech = 4),

PlanDP5 = (SELECT X.PlanCalendar FROM dbo.ex\_Product as X (nolock) WHERE X.dtFirstDay = A.dtFirstDay AND X.NPech = 5),

SlivDP5 = (SELECT X.Sliv FROM dbo.ex\_Product as X (nolock) WHERE X.dtFirstDay = A.dtFirstDay AND X.NPech = 5),

PlanDP6 = (SELECT X.PlanCalendar FROM dbo.ex\_Product as X (nolock) WHERE X.dtFirstDay = A.dtFirstDay AND X.NPech = 6),

SlivDP6 = (SELECT X.Sliv FROM dbo.ex\_Product as X (nolock) WHERE X.dtFirstDay = A.dtFirstDay AND X.NPech = 6),

PlanDP7 = (SELECT X.PlanCalendar FROM dbo.ex\_Product as X (nolock) WHERE X.dtFirstDay = A.dtFirstDay AND X.NPech = 7),

SlivDP7 = (SELECT X.Sliv FROM dbo.ex\_Product as X (nolock) WHERE X.dtFirstDay = A.dtFirstDay AND X.NPech = 7),

PlanDP8 = (SELECT X.PlanCalendar FROM dbo.ex\_Product as X (nolock) WHERE X.dtFirstDay = A.dtFirstDay AND X.NPech = 8),

SlivDP8 = (SELECT X.Sliv FROM dbo.ex\_Product as X (nolock) WHERE X.dtFirstDay = A.dtFirstDay AND X.NPech = 8),

PlanDP9 = (SELECT X.PlanCalendar FROM dbo.ex\_Product as X (nolock) WHERE X.dtFirstDay = A.dtFirstDay AND X.NPech = 9),

SlivDP9 = (SELECT X.Sliv FROM dbo.ex\_Product as X (nolock) WHERE X.dtFirstDay = A.dtFirstDay AND X.NPech = 9),

PlanDP10 = (SELECT X.PlanCalendar FROM dbo.ex\_Product as X (nolock) WHERE X.dtFirstDay = A.dtFirstDay AND X.NPech = 10),

SlivDP10 = (SELECT X.Sliv FROM dbo.ex\_Product as X (nolock) WHERE X.dtFirstDay = A.dtFirstDay AND X.NPech = 10),

PlanDP255 = (SELECT X.PlanCalendar FROM dbo.ex\_Product as X (nolock) WHERE X.dtFirstDay = A.dtFirstDay AND X.NPech = 255),

SlivDP255 = (SELECT X.Sliv FROM dbo.ex\_Product as X (nolock) WHERE X.dtFirstDay = A.dtFirstDay AND X.NPech = 255)

FROM dbo.ex\_Product as A WITH(nolock) WHERE A.dtFirstDay BETWEEN @DateBegYear AND @dtFirstDay

----------------------------------------------------------

-- Всего в натуре c нач.года = сумма Всего в натуре (мес.)

INSERT INTO #Report (

dtFirstDay,

Descr,

PlanDP1, SlivDP1,

PlanDP2, SlivDP2,

PlanDP3, SlivDP3,

PlanDP4, SlivDP4,

PlanDP5, SlivDP5,

PlanDP6, SlivDP6,

PlanDP7, SlivDP7,

PlanDP8, SlivDP8,

PlanDP9, SlivDP9,

PlanDP10, SlivDP10,

PlanDP255, SlivDP255

)

SELECT

@dtFirstDay,

'Всего передельного с начала года',

SUM(X.PlanDP1), SUM(X.SlivDP1),

SUM(X.PlanDP2), SUM(X.SlivDP2),

SUM(X.PlanDP3), SUM(X.SlivDP3),

SUM(X.PlanDP4), SUM(X.SlivDP4),

SUM(X.PlanDP5), SUM(X.SlivDP5),

SUM(X.PlanDP6), SUM(X.SlivDP6),

SUM(X.PlanDP7), SUM(X.SlivDP7),

SUM(X.PlanDP8), SUM(X.SlivDP8),

SUM(X.PlanDP9), SUM(X.SlivDP9),

SUM(X.PlanDP10), SUM(X.SlivDP10),

SUM(X.PlanDP255), SUM(X.SlivDP255)

FROM #Report as X WITH(nolock)

WHERE X.dtFirstDay BETWEEN @DateBegYear AND @dtFirstDay

AND X.Descr in ('Передельный')

SELECT \* FROM #Report

-- Удалить временную таблицу

DROP TABLE #Report

GO

SET QUOTED\_IDENTIFIER OFF

GO

SET ANSI\_NULLS ON

GO

**Пример 5.** Перепишем код хранимой процедуры таким образом, чтобы при реализации была использована временная статическая таблица. Диаграмма взаимодействия объектов при выполнении хранимой процедуры представлена на рис. 1.5.



Рис. 1.5 – Диаграмма взаимодействия выполнения  
хранимой процедуры dbo.ex\_CALC\_Product\_2

Листинг хранимой процедуры **dbo.ex\_CALC\_Product\_2** приведен ниже.

SET QUOTED\_IDENTIFIER ON

GO

SET ANSI\_NULLS OFF

GO

/\*-----------------------------------------------------------

Проект "DBLab"

Пример обработки базы данных MS SQL Server 2008 R2

Разработчик: Лавров Владислав Васильевич

Функции:

Таблица "Производство передельного чугуна"

Расчет производства передельного чугуна с начала года до указанной даты.

При реализации использована временная статическая таблица

Входные данные:

@dtFirstDay - первый день отчетного месяца

Пример вызова:

EXEC dbo.ex\_CALC\_Product\_2 @dtFirstDay='2007.12.01'

\*/-----------------------------------------------------------

CREATE PROC dbo.ex\_CALC\_Product\_2

@dtFirstDay smalldatetime

AS

SET NOCOUNT ON -- отключить сообщения о количестве обработанных записей

SET ANSI\_WARNINGS OFF -- отключить вывод предупреждений

-- Декларировать временную переменную и присвоить дату c начала года

DECLARE @DateBegYear smalldatetime

SET @DateBegYear = Convert(smalldatetime, Convert(varchar, DatePart(year, @dtFirstDay)) + '.01.01')

-- Объявить временную таблицу

DECLARE @Report TABLE (

dtFirstDay smalldatetime NOT NULL,

Descr char(40) NOT NULL,

PlanDP1 real NULL,

SlivDP1 real NULL,

PlanDP2 real NULL,

SlivDP2 real NULL,

PlanDP3 real NULL,

SlivDP3 real NULL,

PlanDP4 real NULL,

SlivDP4 real NULL,

PlanDP5 real NULL,

SlivDP5 real NULL,

PlanDP6 real NULL,

SlivDP6 real NULL,

PlanDP7 real NULL,

SlivDP7 real NULL,

PlanDP8 real NULL,

SlivDP8 real NULL,

PlanDP9 real NULL,

SlivDP9 real NULL,

PlanDP10 real NULL,

SlivDP10 real NULL,

PlanDP255 real NULL,

SlivDP255 real NULL

)

-- Заполнить временную таблицу данными по выплавке передельного чугуна

-- Каждую колонку заполнить запросом к таблице dbo.ex\_Product

INSERT INTO @Report (

dtFirstDay,

Descr,

PlanDP1, SlivDP1,

PlanDP2, SlivDP2,

PlanDP3, SlivDP3,

PlanDP4, SlivDP4,

PlanDP5, SlivDP5,

PlanDP6, SlivDP6,

PlanDP7, SlivDP7,

PlanDP8, SlivDP8,

PlanDP9, SlivDP9,

PlanDP10, SlivDP10,

PlanDP255, SlivDP255

)

SELECT DISTINCT A.dtFirstDay, 'Передельный',

PlanDP1 = (SELECT X.PlanCalendar FROM dbo.ex\_Product as X (nolock) WHERE X.dtFirstDay = A.dtFirstDay AND X.NPech = 1),

SlivDP1 = (SELECT X.Sliv FROM dbo.ex\_Product as X (nolock) WHERE X.dtFirstDay = A.dtFirstDay AND X.NPech = 1),

PlanDP2 = (SELECT X.PlanCalendar FROM dbo.ex\_Product as X (nolock) WHERE X.dtFirstDay = A.dtFirstDay AND X.NPech = 2),

SlivDP2 = (SELECT X.Sliv FROM dbo.ex\_Product as X (nolock) WHERE X.dtFirstDay = A.dtFirstDay AND X.NPech = 2),

PlanDP3 = (SELECT X.PlanCalendar FROM dbo.ex\_Product as X (nolock) WHERE X.dtFirstDay = A.dtFirstDay AND X.NPech = 3),

SlivDP3 = (SELECT X.Sliv FROM dbo.ex\_Product as X (nolock) WHERE X.dtFirstDay = A.dtFirstDay AND X.NPech = 3),

PlanDP4 = (SELECT X.PlanCalendar FROM dbo.ex\_Product as X (nolock) WHERE X.dtFirstDay = A.dtFirstDay AND X.NPech = 4),

SlivDP4 = (SELECT X.Sliv FROM dbo.ex\_Product as X (nolock) WHERE X.dtFirstDay = A.dtFirstDay AND X.NPech = 4),

PlanDP5 = (SELECT X.PlanCalendar FROM dbo.ex\_Product as X (nolock) WHERE X.dtFirstDay = A.dtFirstDay AND X.NPech = 5),

SlivDP5 = (SELECT X.Sliv FROM dbo.ex\_Product as X (nolock) WHERE X.dtFirstDay = A.dtFirstDay AND X.NPech = 5),

PlanDP6 = (SELECT X.PlanCalendar FROM dbo.ex\_Product as X (nolock) WHERE X.dtFirstDay = A.dtFirstDay AND X.NPech = 6),

SlivDP6 = (SELECT X.Sliv FROM dbo.ex\_Product as X (nolock) WHERE X.dtFirstDay = A.dtFirstDay AND X.NPech = 6),

PlanDP7 = (SELECT X.PlanCalendar FROM dbo.ex\_Product as X (nolock) WHERE X.dtFirstDay = A.dtFirstDay AND X.NPech = 7),

SlivDP7 = (SELECT X.Sliv FROM dbo.ex\_Product as X (nolock) WHERE X.dtFirstDay = A.dtFirstDay AND X.NPech = 7),

PlanDP8 = (SELECT X.PlanCalendar FROM dbo.ex\_Product as X (nolock) WHERE X.dtFirstDay = A.dtFirstDay AND X.NPech = 8),

SlivDP8 = (SELECT X.Sliv FROM dbo.ex\_Product as X (nolock) WHERE X.dtFirstDay = A.dtFirstDay AND X.NPech = 8),

PlanDP9 = (SELECT X.PlanCalendar FROM dbo.ex\_Product as X (nolock) WHERE X.dtFirstDay = A.dtFirstDay AND X.NPech = 9),

SlivDP9 = (SELECT X.Sliv FROM dbo.ex\_Product as X (nolock) WHERE X.dtFirstDay = A.dtFirstDay AND X.NPech = 9),

PlanDP10 = (SELECT X.PlanCalendar FROM dbo.ex\_Product as X (nolock) WHERE X.dtFirstDay = A.dtFirstDay AND X.NPech = 10),

SlivDP10 = (SELECT X.Sliv FROM dbo.ex\_Product as X (nolock) WHERE X.dtFirstDay = A.dtFirstDay AND X.NPech = 10),

PlanDP255 = (SELECT X.PlanCalendar FROM dbo.ex\_Product as X (nolock) WHERE X.dtFirstDay = A.dtFirstDay AND X.NPech = 255),

SlivDP255 = (SELECT X.Sliv FROM dbo.ex\_Product as X (nolock) WHERE X.dtFirstDay = A.dtFirstDay AND X.NPech = 255)

FROM dbo.ex\_Product as A WITH(nolock) WHERE A.dtFirstDay BETWEEN @DateBegYear AND @dtFirstDay

-----------------------------------------------------------------------

-- Всего в натуре c нач.года = сумма Всего в натуре (мес.)

INSERT INTO @Report (

dtFirstDay,

Descr,

PlanDP1, SlivDP1,

PlanDP2, SlivDP2,

PlanDP3, SlivDP3,

PlanDP4, SlivDP4,

PlanDP5, SlivDP5,

PlanDP6, SlivDP6,

PlanDP7, SlivDP7,

PlanDP8, SlivDP8,

PlanDP9, SlivDP9,

PlanDP10, SlivDP10,

PlanDP255, SlivDP255

)

SELECT

@dtFirstDay,

'Всего передельного с начала года',

SUM(X.PlanDP1), SUM(X.SlivDP1),

SUM(X.PlanDP2), SUM(X.SlivDP2),

SUM(X.PlanDP3), SUM(X.SlivDP3),

SUM(X.PlanDP4), SUM(X.SlivDP4),

SUM(X.PlanDP5), SUM(X.SlivDP5),

SUM(X.PlanDP6), SUM(X.SlivDP6),

SUM(X.PlanDP7), SUM(X.SlivDP7),

SUM(X.PlanDP8), SUM(X.SlivDP8),

SUM(X.PlanDP9), SUM(X.SlivDP9),

SUM(X.PlanDP10), SUM(X.SlivDP10),

SUM(X.PlanDP255), SUM(X.SlivDP255)

FROM @Report as X

WHERE X.dtFirstDay BETWEEN @DateBegYear AND @dtFirstDay

AND X.Descr in ('Передельный')

SELECT \* FROM @Report

GO

SET QUOTED\_IDENTIFIER OFF

GO

SET ANSI\_NULLS ON

GO

Как видно из результатов выполнения представленной выше процедуры, на экране появляются столбцы для печей, которых не существует. Чтобы этого избежать, необходимо в ходе выполнения определять количество работающих печей и формировать структуру таблицы по производству только для них.

**Пример 6.** Одним из подходов к реализации программного кода является использование временной таблицы с переменной структурой. Диаграмма взаимодействия объектов при выполнении хранимой процедуры представлена на рис. 1.6.



Рис. 1.6 – Диаграмма взаимодействия выполнения   
хранимой процедуры dbo.ex\_CALC\_Product\_3

Листинг хранимой процедуры **dbo.ex\_CALC\_Product\_3** приведен ниже.

SET QUOTED\_IDENTIFIER ON

GO

SET ANSI\_NULLS OFF

GO

/\*-----------------------------------------------------------

Проект "DBLab"

Пример обработки базы данных MS SQL Server 2008 R2

Разработчик: Лавров Владислав Васильевич

Функции:

Таблица "Производство передельного чугуна"

Расчет производства передельного чугуна за отчетный месяц.

При реализации использована временная таблица,

структура которой меняется в ходе выполнения процедуры

Входные данные:

@dtFirstDay - первый день отчетного месяца

@debug tinyint - параметр для отладки (0 - рабочий по умолчанию, 1 - отладка)

Пример вызова:

EXEC dbo.ex\_CALC\_Product\_3 @dtFirstDay='2007.02.01'

\*/-----------------------------------------------------------

CREATE PROC dbo.ex\_CALC\_Product\_3

@dtFirstDay smalldatetime,

@debug tinyint = 0 -- параметр для отладки (0 - рабочий по умолчанию, 1 - отладка)

AS

SET NOCOUNT ON -- отключить сообщения о количестве обработанных записей

SET ANSI\_WARNINGS OFF -- отключить вывод предупреждений

-- Создать временную таблицу. Впоследствии потребуется динамически добавить столбцы

CREATE TABLE #Report

(

dtFirstDay smalldatetime NOT NULL,

IdDescr int identity(1,1) NOT NULL,

Descr char(40) NOT NULL

)

-- Объявить таблицу по печам, структуру менять не будем.

DECLARE @ListAllPech TABLE

(

IdRow tinyint Identity(1,1) NOT NULL,

NPech tinyint NOT NULL,

sNPech varchar(3) NOT NULL

)

-- Объявить временные переменные

Declare @sNPech varchar( 3),

@sDT varchar(10),

@maxIdRow int,

@IdRow int,

@sFields varchar(8000),

@nameTable sysname

SET @sDT = Convert(varchar(10), @dtFirstDay, 102) -- сформировать формат даты

SET @nameTable = 'dbo.ex\_Product'

-- Найти все рабочие печи с начала года по текущий месяц и вставить в таблицу

INSERT INTO @ListAllPech (NPech, sNPech)

SELECT DISTINCT P.NPech, LTrim(Str(P.NPech))

FROM dbo.ex\_Product as P WITH(nolock)

WHERE P.dtFirstDay = @dtFirstDay

ORDER BY NPech

-- Вставить строку во временную таблицу

INSERT INTO #Report (dtFirstDay, Descr)

VALUES (@sDT, 'Передельный чугун, т')

-- Подготовить переменные для циклической обработки

SET @maxIdRow = (SELECT MAX(IdRow) FROM @ListAllPech)

SET @IdRow = 1

-- Список полей, которые вывести в отчете

SET @sFields='Descr'

-- Начать цикл заполнения данными

WHILE @IdRow <= @maxIdRow

BEGIN

SET @sNPech = NULL

SET @sNPech = (SELECT sNPech FROM @ListAllPech WHERE IdRow=@IdRow)

IF @sNPech IS NOT NULL

begin

-- Сформировать число столбцов, в зависимости от количества работающих печей

EXEC('ALTER TABLE #Report' +

' ADD PlanDP' + @sNPech + ' decimal(12,3) NULL'

)

EXEC('ALTER TABLE #Report' +

' ADD SlivDP' + @sNPech + ' decimal(12,3) NULL'

)

-- Заполнить столбцы данными

EXEC('UPDATE #Report ' +

' SET PlanDP' + @sNPech + '=' +

' ( SELECT PlanCalendar' +

' FROM '+ @nameTable + ' WITH(nolock)' +

' WHERE dtFirstDay = ''' + @sDT + ''' AND NPech =' + @sNPech + ')'

)

EXEC('UPDATE #Report ' +

' SET SlivDP' + @sNPech + '=' +

' ( SELECT Sliv' +

' FROM '+ @nameTable + ' WITH(nolock)' +

' WHERE dtFirstDay = ''' + @sDT + ''' AND NPech =' + @sNPech + ')'

)

-- Отобразить все числа в нужном формате

SET @sFields = @sFields +

',ISNULL( Convert(char(9), Str(PlanDP' + @sNPech + ', 9, 2)), Space(9)) as PlanDP' + @sNPech +

',ISNULL( Convert(char(9), Str(SlivDP' + @sNPech + ', 9, 2)), Space(9)) as SlivDP' + @sNPech

end -- Окончание блока обработки данных

SET @IdRow = @IdRow + 1

END -- Окончание цикла

-- Если установлен рабочий режим, то показать нужные столбцы из временной таблицы

if @debug = 0

begin

EXEC ('SELECT ' + @sFields + ' FROM #Report WHERE dtFirstDay =''' + @sDT + '''')

end

-- Иначе установлен тестовый режим, надо показать всю таблицу

else

begin

SELECT \* FROM #Report

end

-- Удалить временную таблицу

DROP TABLE #Report

GO

SET QUOTED\_IDENTIFIER OFF

GO

SET ANSI\_NULLS ON

GO

**Задание на самостоятельную работу.** Необходимо усовершенствовать функциональность вышеприведенной процедуры с тем, чтобы в результате отображались данные по производству передельного чугуна на каждой доменной печи с начала года (с накоплением). Другими словами, необходимо добавить строку «Передельный чугун с начала года, т». Построить диаграмму взаимодействия.

**Пример 7.** Более перспективным с точки зрения производительности выполнения запросов является использование постоянных таблиц, которые заполняют данными хранимые процедуры для последующей обработки другими хранимыми процедурами. Диаграмма взаимодействия объектов при выполнении хранимой процедуры представлена на рис. 1.7.



Рис. 1.7 – Диаграмма взаимодействия выполнения  
хранимой процедуры dbo.ex\_Report\_Product

Для примера, создадим такую таблицу для временного хранения данных по производству передельного чугуна. Необходимо выполнить представленный ниже скрипт для создания в базе данных DBLab таблицы **dbo.ex\_arrayProduct**.

if exists (select \* from dbo.sysobjects where id = object\_id(N'[dbo].[ex\_arrayProduct]') and OBJECTPROPERTY(id, N'IsUserTable') = 1)

drop table [dbo].[ex\_arrayProduct]

GO

CREATE TABLE [dbo].[ex\_arrayProduct] (

[SPID] [smallint] NOT NULL ,

[dtFirstDay] [smalldatetime] NOT NULL ,

[IdDescr] [int] IDENTITY (1, 1) NOT NULL ,

[Descr] [varchar] (50) COLLATE Cyrillic\_General\_CI\_AS NOT NULL ,

[Mnemo] [varchar] (50) COLLATE Cyrillic\_General\_CI\_AS NOT NULL ,

[PlanDP1] [real] NULL ,

[SlivDP1] [real] NULL ,

[PercentDP1] AS (convert(decimal(5,2),round(case when ([PlanDP1] > 0) then ([SlivDP1] / [PlanDP1] \* 100.0) end,2))) ,

[PlanDP2] [real] NULL ,

[SlivDP2] [real] NULL ,

[PercentDP2] AS (convert(decimal(5,2),round(case when ([PlanDP2] > 0) then ([SlivDP2] / [PlanDP2] \* 100.0) end,2))) ,

[PlanDP3] [real] NULL ,

[SlivDP3] [real] NULL ,

[PercentDP3] AS (convert(decimal(5,2),round(case when ([PlanDP3] > 0) then ([SlivDP3] / [PlanDP3] \* 100.0) end,2))) ,

[PlanDP4] [real] NULL ,

[SlivDP4] [real] NULL ,

[PercentDP4] AS (convert(decimal(5,2),round(case when ([PlanDP4] > 0) then ([SlivDP4] / [PlanDP4] \* 100.0) end,2))) ,

[PlanDP5] [real] NULL ,

[SlivDP5] [real] NULL ,

[PercentDP5] AS (convert(decimal(5,2),round(case when ([PlanDP5] > 0) then ([SlivDP5] / [PlanDP5] \* 100.0) end,2))) ,

[PlanDP6] [real] NULL ,

[SlivDP6] [real] NULL ,

[PercentDP6] AS (convert(decimal(5,2),round(case when ([PlanDP6] > 0) then ([SlivDP6] / [PlanDP6] \* 100.0) end,2))) ,

[PlanDP7] [real] NULL ,

[SlivDP7] [real] NULL ,

[PercentDP7] AS (convert(decimal(5,2),round(case when ([PlanDP7] > 0) then ([SlivDP7] / [PlanDP7] \* 100.0) end,2))) ,

[PlanDP8] [real] NULL ,

[SlivDP8] [real] NULL ,

[PercentDP8] AS (convert(decimal(5,2),round(case when ([PlanDP8] > 0) then ([SlivDP8] / [PlanDP8] \* 100.0) end,2))) ,

[PlanDP9] [real] NULL ,

[SlivDP9] [real] NULL ,

[PercentDP9] AS (convert(decimal(5,2),round(case when ([PlanDP9] > 0) then ([SlivDP9] / [PlanDP9] \* 100.0) end,2))) ,

[PlanDP10] [real] NULL ,

[SlivDP10] [real] NULL ,

[PercentDP10] AS (convert(decimal(5,2),round(case when ([PlanDP10] > 0) then ([SlivDP10] / [PlanDP10] \* 100.0) end,2))) ,

[PlanDP255] [real] NULL ,

[SlivDP255] [real] NULL ,

[PercentDP255] AS (convert(decimal(5,2),round(case when ([PlanDP255] > 0) then ([SlivDP255] / [PlanDP255] \* 100.0) end,2)))

) ON [PRIMARY]

GO

Атрибут таблицы SPID предназначен для хранения номера системного процесса, который производит заполнение таблицы данными.

Тогда заполнение данных по производству передельного чугуна можно производить в отдельной хранимой процедуре и вызывать ее по мере необходимости. Листинг такой хранимой процедуры **dbo.ex\_CALC\_Product\_4** представлен ниже. Обратите внимание на то, что перед началом все данные для текущего системного процесса в таблице стираются. Номер текущего системного процесса определяется с использованием встроенной системной переменной @@SPID.

SET QUOTED\_IDENTIFIER ON

GO

SET ANSI\_NULLS OFF

GO

/\*-----------------------------------------------------------

Проект "DBLab"

Пример обработки базы данных MS SQL Server 2008 R2

Разработчик: Лавров Владислав Васильевич

Функции:

Таблица "Производство передельного чугуна"

Расчет производства передельного чугуна с начала года до указанной даты.

При реализации использована постоянная таблица

Входные данные:

@dtFirstDay - первый день отчетного месяца

Пример вызова:

EXEC dbo.ex\_CALC\_Product\_4 @dtFirstDay='2007.12.01'

\*/-----------------------------------------------------------

CREATE PROC dbo.ex\_CALC\_Product\_4

@dtFirstDay smalldatetime

AS

SET NOCOUNT ON -- отключить сообщения о количестве обработанных записей

SET ANSI\_WARNINGS OFF -- отключить вывод предупреждений

-- Декларировать временную переменную и присвоить дату c начала года

DECLARE @DateBegYear smalldatetime

SET @DateBegYear = Convert(smalldatetime, Convert(varchar, DatePart(year, @dtFirstDay)) + '.01.01')

-----------------------------------------------------------

-- Удалить все данные в таблице для текущего системного процесса

DELETE FROM dbo.ex\_arrayProduct WHERE SPID=@@SPID

-----------------------------------------------------------

-- Заполнить временную таблицу данными по выплавке передельного чугуна

-- Каждую колонку заполнить запросом к таблице dbo.ex\_Product

INSERT INTO dbo.ex\_arrayProduct (

SPID,

dtFirstDay,

Descr,

Mnemo,

PlanDP1, SlivDP1,

PlanDP2, SlivDP2,

PlanDP3, SlivDP3,

PlanDP4, SlivDP4,

PlanDP5, SlivDP5,

PlanDP6, SlivDP6,

PlanDP7, SlivDP7,

PlanDP8, SlivDP8,

PlanDP9, SlivDP9,

PlanDP10, SlivDP10,

PlanDP255, SlivDP255

)

SELECT DISTINCT @@SPID, A.dtFirstDay, 'Передельный', 'Передельный',

PlanDP1 = (SELECT X.PlanCalendar FROM dbo.ex\_Product as X (nolock) WHERE X.dtFirstDay = A.dtFirstDay AND X.NPech = 1),

SlivDP1 = (SELECT X.Sliv FROM dbo.ex\_Product as X (nolock) WHERE X.dtFirstDay = A.dtFirstDay AND X.NPech = 1),

PlanDP2 = (SELECT X.PlanCalendar FROM dbo.ex\_Product as X (nolock) WHERE X.dtFirstDay = A.dtFirstDay AND X.NPech = 2),

SlivDP2 = (SELECT X.Sliv FROM dbo.ex\_Product as X (nolock) WHERE X.dtFirstDay = A.dtFirstDay AND X.NPech = 2),

PlanDP3 = (SELECT X.PlanCalendar FROM dbo.ex\_Product as X (nolock) WHERE X.dtFirstDay = A.dtFirstDay AND X.NPech = 3),

SlivDP3 = (SELECT X.Sliv FROM dbo.ex\_Product as X (nolock) WHERE X.dtFirstDay = A.dtFirstDay AND X.NPech = 3),

PlanDP4 = (SELECT X.PlanCalendar FROM dbo.ex\_Product as X (nolock) WHERE X.dtFirstDay = A.dtFirstDay AND X.NPech = 4),

SlivDP4 = (SELECT X.Sliv FROM dbo.ex\_Product as X (nolock) WHERE X.dtFirstDay = A.dtFirstDay AND X.NPech = 4),

PlanDP5 = (SELECT X.PlanCalendar FROM dbo.ex\_Product as X (nolock) WHERE X.dtFirstDay = A.dtFirstDay AND X.NPech = 5),

SlivDP5 = (SELECT X.Sliv FROM dbo.ex\_Product as X (nolock) WHERE X.dtFirstDay = A.dtFirstDay AND X.NPech = 5),

PlanDP6 = (SELECT X.PlanCalendar FROM dbo.ex\_Product as X (nolock) WHERE X.dtFirstDay = A.dtFirstDay AND X.NPech = 6),

SlivDP6 = (SELECT X.Sliv FROM dbo.ex\_Product as X (nolock) WHERE X.dtFirstDay = A.dtFirstDay AND X.NPech = 6),

PlanDP7 = (SELECT X.PlanCalendar FROM dbo.ex\_Product as X (nolock) WHERE X.dtFirstDay = A.dtFirstDay AND X.NPech = 7),

SlivDP7 = (SELECT X.Sliv FROM dbo.ex\_Product as X (nolock) WHERE X.dtFirstDay = A.dtFirstDay AND X.NPech = 7),

PlanDP8 = (SELECT X.PlanCalendar FROM dbo.ex\_Product as X (nolock) WHERE X.dtFirstDay = A.dtFirstDay AND X.NPech = 8),

SlivDP8 = (SELECT X.Sliv FROM dbo.ex\_Product as X (nolock) WHERE X.dtFirstDay = A.dtFirstDay AND X.NPech = 8),

PlanDP9 = (SELECT X.PlanCalendar FROM dbo.ex\_Product as X (nolock) WHERE X.dtFirstDay = A.dtFirstDay AND X.NPech = 9),

SlivDP9 = (SELECT X.Sliv FROM dbo.ex\_Product as X (nolock) WHERE X.dtFirstDay = A.dtFirstDay AND X.NPech = 9),

PlanDP10 = (SELECT X.PlanCalendar FROM dbo.ex\_Product as X (nolock) WHERE X.dtFirstDay = A.dtFirstDay AND X.NPech = 10),

SlivDP10 = (SELECT X.Sliv FROM dbo.ex\_Product as X (nolock) WHERE X.dtFirstDay = A.dtFirstDay AND X.NPech = 10),

PlanDP255 = (SELECT X.PlanCalendar FROM dbo.ex\_Product as X (nolock) WHERE X.dtFirstDay = A.dtFirstDay AND X.NPech = 255),

SlivDP255 = (SELECT X.Sliv FROM dbo.ex\_Product as X (nolock) WHERE X.dtFirstDay = A.dtFirstDay AND X.NPech = 255)

FROM dbo.ex\_Product as A WITH(nolock) WHERE A.dtFirstDay BETWEEN @DateBegYear AND @dtFirstDay

-- Всего передельного чугуна c начала года = сумма Передельный (мес.)

INSERT INTO dbo.ex\_arrayProduct (

SPID,

dtFirstDay,

Descr,

Mnemo,

PlanDP1, SlivDP1,

PlanDP2, SlivDP2,

PlanDP3, SlivDP3,

PlanDP4, SlivDP4,

PlanDP5, SlivDP5,

PlanDP6, SlivDP6,

PlanDP7, SlivDP7,

PlanDP8, SlivDP8,

PlanDP9, SlivDP9,

PlanDP10, SlivDP10,

PlanDP255, SlivDP255

)

SELECT @@SPID, X.dtFirstDay,

'Всего передельного с начала года',

'Всего передельного с начала года',

SUM(X.PlanDP1), SUM(X.SlivDP1),

SUM(X.PlanDP2), SUM(X.SlivDP2),

SUM(X.PlanDP3), SUM(X.SlivDP3),

SUM(X.PlanDP4), SUM(X.SlivDP4),

SUM(X.PlanDP5), SUM(X.SlivDP5),

SUM(X.PlanDP6), SUM(X.SlivDP6),

SUM(X.PlanDP7), SUM(X.SlivDP7),

SUM(X.PlanDP8), SUM(X.SlivDP8),

SUM(X.PlanDP9), SUM(X.SlivDP9),

SUM(X.PlanDP10), SUM(X.SlivDP10),

SUM(X.PlanDP255), SUM(X.SlivDP255)

FROM dbo.ex\_arrayProduct as X WITH(nolock)

WHERE X.SPID=@@SPID AND X.dtFirstDay BETWEEN @DateBegYear AND @dtFirstDay

AND X.Mnemo in ('Передельный')

GROUP BY X.dtFirstDay

--SELECT \* FROM dbo.ex\_arrayProduct

GO

SET QUOTED\_IDENTIFIER OFF

GO

SET ANSI\_NULLS ON

GO

Приведенная выше хранимая процедура ничего не выводит, она заполняет данными таблицу dbo.ex\_arrayProduct в базе данных DBLab. В последствии эти данные могут быть использованы, например, для формирование отчета по производству передельного чугуна. Листинг хранимой процедуры **dbo.ex\_Report\_Product** представлен ниже.

SET QUOTED\_IDENTIFIER ON

GO

SET ANSI\_NULLS OFF

GO

/\*-----------------------------------------------------------

Проект "ExampleDB"

Пример обработки базы данных в среде Visual Studio 2010 .NET"

Разработчик: Лавров Владислав Васильевич

Функции:

Формирование отчета "Производство передельного чугуна"

Входные данные:

@dtFirstDay - первый день отчетной даты

@debug - параметр для отладки (0 - рабочий по умолчанию, 1 - отладка)

Пример вызова:

EXEC dbo.ex\_Report\_Product @dtFirstDay='2007.01.01'

\*/-----------------------------------------------------------

CREATE PROC dbo.ex\_Report\_Product

@dtFirstDay smalldatetime,

@debug tinyint = 0 -- параметр для отладки (0 - рабочий по умолчанию, 1 - отладка)

as

SET NOCOUNT ON -- отключить сообщения о количестве обработанных записей

SET ANSI\_WARNINGS OFF -- отключить вывод предупреждений

-- Создать временную таблицу. Впоследствии потребуется динамически добавить столбцы

CREATE TABLE #Report

(

dtFirstDay smalldatetime NOT NULL,

IdDescr int identity(1,1) NOT NULL,

Descr char(60) NOT NULL,

Mnemo varchar(60) NOT NULL

)

-- Объявить таблицу по печам, структуру менять не будем.

DECLARE @ListAllPech TABLE

(

IdRow tinyint Identity(1,1) NOT NULL,

NPech tinyint NOT NULL,

sNPech varchar(3) NOT NULL

)

-- Объявить временные переменные

Declare @sNPech varchar( 3),

@sDT varchar(10),

@maxIdRow int,

@IdRow int,

@sFields varchar(8000),

@nameTable sysname

SET @sDT = Convert(varchar(10), @dtFirstDay, 102) -- сформировать формат даты

SET @nameTable = 'dbo.ex\_arrayProduct'

-- SET @nameTable = 'dbo.ex\_Product'

-- Расчет продукции доменного цеха

EXEC dbo.ex\_CALC\_Product\_4 @dtFirstDay=@dtFirstDay

-- <<<<<<<< Формировать отчет >>>>>>>>>>>

-- Найти все рабочие печи с начала года по текущий месяц и вставить в таблицу

INSERT INTO @ListAllPech (NPech, sNPech)

SELECT P.NPech, LTrim(Str(P.NPech))

FROM dbo.ex\_Product as P WITH(nolock)

WHERE P.dtFirstDay = @dtFirstDay

ORDER BY NPech

-- Подготовить строки во временной таблице

INSERT INTO #Report (dtFirstDay, Descr, Mnemo)

VALUES (@dtFirstDay, 'План по передельному чугуну (календарный), т', 'Передельный')

INSERT INTO #Report (dtFirstDay, Descr, Mnemo)

VALUES (@dtFirstDay, 'Слив передельного чистого чугуна, т', 'Передельный')

INSERT INTO #Report (dtFirstDay, Descr, Mnemo)

VALUES (@dtFirstDay, 'Процент выполненного производства по передельному чугуну, т', 'Передельный')

-- Подготовить переменные для циклической обработки

SET @maxIdRow = (SELECT MAX(IdRow) FROM @ListAllPech)

SET @IdRow = 1

-- Список полей, которые вывести в отчете

SET @sFields='Descr'

-- Начать цикл заполнения данными

WHILE @IdRow <= @maxIdRow

BEGIN

SET @sNPech = NULL

SET @sNPech = (SELECT sNPech FROM @ListAllPech WHERE IdRow=@IdRow)

IF @sNPech IS NOT NULL

begin

-- Сформировать число столбцов, в зависимости от количества работающих печей

EXEC('ALTER TABLE #Report' +

' ADD DP' + @sNPech + ' decimal(12,3) NULL'

)

-- Заполнить столбцы данными

EXEC('UPDATE R' +

' SET DP' + @sNPech + ' =T.PlanDP' + @sNPech +

' FROM #Report as R' +

' INNER JOIN ' + @nameTable + ' as T' +

' ON R.dtFirstDay=T.dtFirstDay AND R.Mnemo=T.Mnemo' +

' WHERE T.SPID=@@SPID ' +

' AND R.Descr= ''План по передельному чугуну (календарный), т'''

)

-- Заполнить столбцы данными

EXEC('UPDATE R' +

' SET DP' + @sNPech + ' =T.SlivDP' + @sNPech +

' FROM #Report as R' +

' INNER JOIN ' + @nameTable + ' as T' +

' ON R.dtFirstDay=T.dtFirstDay AND R.Mnemo=T.Mnemo' +

' WHERE T.SPID=@@SPID ' +

' AND R.Descr= ''Слив передельного чистого чугуна, т'''

)

-- Заполнить столбцы данными

EXEC('UPDATE R' +

' SET DP' + @sNPech + ' =T.PercentDP' + @sNPech +

' FROM #Report as R' +

' INNER JOIN ' + @nameTable + ' as T' +

' ON R.dtFirstDay=T.dtFirstDay AND R.Mnemo=T.Mnemo' +

' WHERE T.SPID=@@SPID ' +

' AND R.Descr= ''Процент выполненного производства по передельному чугуну, т'''

)

-- Отобразить все числа в нужном формате

SET @sFields=@sFields +

',ISNULL( Convert(char(9), Str(DP' + @sNPech +',9, 2)), Space(9) ) as DP' + @sNPech

end -- Окончание блока обработки данных

SET @IdRow = @IdRow + 1

END -- Окончание цикла

-- Если установлен рабочий режим, то показать нужные столбцы из временной таблицы

if @debug = 0

begin

EXEC ('SELECT ' + @sFields + ' FROM #Report WHERE dtFirstDay =''' + @sDT + '''')

end

-- Иначе установлен тестовый режим, надо показать всю таблицу

else

begin

SELECT \* FROM #Report

end

-- Удалить временную таблицу

DROP TABLE #Report

SET ANSI\_WARNINGS ON -- включить вывод предупреждений

SET NOCOUNT OFF -- передавать сообщения о количестве обработанных записей

GO

SET QUOTED\_IDENTIFIER OFF

GO

SET ANSI\_NULLS ON

GO

**2 Создание триггеров**

Триггеры – инструмент SQL-сервера, используемый для поддержания целостности данных в базе. На практике триггеры представляют собой команды, которые могут выполняться при определенных действиях в таблице базы данных, например, добавлении и удалении записей.

Существует три типа триггеров, классифицируемых в зависимости от области их использования: при добавлении (insert), изменении (update) и удалении (delete).

**Пример 8.** Для примера создадим базу данных DBLab\_Triggers и наполним ее данными из файла электронных таблиц DataSource\_Triggers.xls. База данных содержит две таблицы, структура которых показана ниже

**Т\_Ресурсы**

|  |  |  |
| --- | --- | --- |
| **Имя поля** | **Тип поля** | **Примечание** |
| КодРесурса | tinyint | PK |
| НаименованиеРесурса | varchar(20) |  |
| ОстатокРесурса | real |  |

**Т\_Процесс**

|  |  |  |
| --- | --- | --- |
| Имя поля | Тип поля | Примечание |
| КодАгрегата | tinyint | PK |
| КодРесурса | tinyint | PK |
| ДатаПроведенияПроцесса | smalldatetime | PK |
| РасходРесурса | real |  |

Необходимо определить триггер Verify\_Ostatok для таблицы Т\_Процесс, который будет проверять величину вводимого значения в поле РасходРесурса на предмет превышения остатка ресурсов в таблице Т\_Ресурсы. В случае если вводимое значение не превышает остаток для соответствующего вида ресурсов – величину в поле ОстатокРесурса, – тогда производимая операция вставки считается успешной, а в таблице Т\_Ресурсы величина поля ОстатокРесурса для этого ресурса уменьшается на вводимую величину. Тем самым, создаваемый триггер не позволит ввести данные в таблицу Т\_Процесс, если вводимый расход ресурса будет превышать остаток в таблице Т\_Ресурсы.

Для создания триггера Verify\_Ostatok лучше использовать запрос, скрипт которого представлен ниже. Выполнять этот запрос надо, естественно, к базе данных DBLab\_Triggers.

-------------------------------------------------------------

/\*

Создание триггера Verify\_Ostatok для вставки данных в таблицу Т\_Процесс

Функция триггера:

Проверка величины вводимого значения в поле РасходРесурса на предмет превышения

остатка ресурсов в таблице Т\_Ресурсы.

В случае если вводимое значение не превышает остаток для соответствующего вида ресурсов

– величину в поле ОстатокРесурса, – тогда производимая операция вставки считается успешной,

а в таблице Т\_Ресурсы величина поля ОстатокРесурса для этого ресурса уменьшается на вводимую величину.

Автор: Лавров Владислав Васильевич

\*/

-------------------------------------------------------------

CREATE TRIGGER Verify\_Ostatok ON Т\_Процесс

FOR INSERT

AS

DECLARE @Sum float, -- остаток ресурса

@Kod int -- код ресурса

SELECT @Sum = I.РасходРесурса,

@Kod= I.КодРесурса

FROM Т\_Ресурсы A, Inserted I

WHERE A.КодРесурса = I.КодРесурса

IF NOT EXISTS (SELECT \* FROM Т\_Ресурсы

WHERE ОстатокРесурса >= @Sum and КодРесурса = @Kod)

BEGIN

ROLLBACK TRAN

RAISERROR ('Данного ресурса нет!',16,10)

END

ELSE

BEGIN

UPDATE Т\_Ресурсы

SET ОстатокРесурса = ОстатокРесурса - @Sum

WHERE КодРесурса = @Kod

END

Рассмотрим подробнее команды данного листинга. Здесь с помощью первых строк определяется создание нового триггера с именем Verify\_Ostatok для таблицы Т\_Процесс, причем создание триггера определяется для процесса добавления новых записей (ключевое слово INSERT). Далее определяются переменные @Sum, которой присваивается значение поля РасходРесурса добавляемой записи, а также @Kod, которой присваивается значение поля КодРесурса. В процессе использования триггера создаются соответствующие временные таблицы INSERTED, UPDATED и DELETED, хранящие в себе соответственно добавляемые, изменяемые и удаляемые значения начальной таблицы Т\_Процесс. С помощью представленного оператора SELECT переменной @Sum присваивается значение поля I.РасходРесурса из таблицы Т\_Процесс, т.е. значение поля РасходРесурса вновь добавляемой записи, а переменной @Kod – соответственно значение поля I.КодРесурса.

Следующим шагом работы триггера будет изменение значения поля ОстатокРесурса таблицы Т\_Ресурсы на вводимую величину, которая хранится в переменной @Sum. Изменения в таблице Т\_Ресурсы производятся для соответствующего кода ресурса, т.е. КодРесурса=@Kod.

Далее проверяем величину остатка ресурса в таблице Т\_Ресурсы, для чего поле ОстатокРесурса сравниваем с переменной @Sum. Если данная проверка производится успешно, то выполняется блок операторов, заключенных в области BEGIN…END. Здесь с помощью команды ROLLBACK TRAN, используемой при работе с транзакциями, осуществляется откат последних операций – операции добавления записи и внесения изменений в таблицу Т\_Ресурсы. При этом с помощью оператора RAISERROR осуществляется выдача системного сообщения об ошибке с текстом, указанным внутри данного оператора. Значения 16 и 10 данной команды определяют уроень критичности ошибки.

**3 Задание для самостоятельной работы**

**Задание 1.** Написать хранимые процедуры для сопровождения таблицы по учету расхода кокса на доменных печах (dbo.ex\_SEL\_Koks, dbo.ex\_INS\_Koks, dbo.ex\_DEL\_Koks, dbo.ex\_UPD\_Koks).

**Задание 2.** Написать хранимую процедуру, которая производит расчет расхода кокса на доменных печах за произвольный месяц, а также за период с начала года до указанного месяца. Дату начала отчетного месяца определить как входной параметр процедуры. В результате сформировать следующую таблицу

|  |  |  |  |
| --- | --- | --- | --- |
| Показатель | Влажный скиповый кокс, т | Сухой скиповый кокс, т | Сухая коксовая мелочь, т |
| ДП-1 |  |  |  |
| … | … | … | … |
| ДП-10 |  |  |  |
| Цех |  |  |  |

**Задание 3.** Написать хранимую процедуру, которая формирует отчет по использованию кокса на доменных печах в отчетном месяце в виде следующей таблицы. Дату начала отчетного месяца определить как входной параметр процедуры.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Показатель | ДП-1 | ... | ДП-10 | Цех |
| Влажный скиповый кокс, т |  | … |  |  |
| Сухой скиповый кокс, т |  | … |  |  |
| Сухая коксовая мелочь, т |  | … |  |  |

Список использованных источников

1. Хансен Г., Хансен Д. Базы данных и управление. - М.: Бином, 2014.
2. Дейт К. Введение в системы баз данных. - М.: "Вильямс", 2012.
3. Конноли Т., Бегг К., Страчан А. Базы данных. Проектирование, реализация и сопровождение. - М.- С./П.- К., 2013.
4. Грофф Д.Р., Вайнберг П.Н. SQL: Полное руководство. - Киев: BMV, "Ирина", 2011.
5. Когаловский М.Р. Энциклопедия технологий баз данных. - М.: Финансы и статистика, 2013.
6. Григорьев Ю.А., Ревунков Г.И. Банки данных. - М.: МГТУ им. Баумана, 2012.
7. Уорсли Дж., Дрейк Дж. PostgreSQL. - Санкт-Петербург, Питер, - 2013.
8. Саймон А. Когаловский М.Р. Стратегические технологии баз данных: менеджмент на 2000 год. - М.: Бином, 2012.
9. Грабер М. Введение в SQL. - М.: ЛОРИ, 2013.
10. Барсегян А.А.,Куприянов М.С.,Степаненко В.В.,Холод И.И. Методы и модели анализа данных: OLAP и Data Mining. - Санкт-Петербург, БХВ-Петербург, 2014.
11. Хоменко А.Д., Цыганков В.М., Мальцев М.Г. Базы данных. - Санкт-Петербург, Корона, 2012.
12. Корнеев В.В., Гарев А. Ф., Васютин С.В., Райх В.В. Базы данных. Интеллектуальная обработка информации. - М. - Нолидж, 2013.