

нестандартных значений для ее характеристик, то для этого можно использовать и диалоговые средства Management Studio.

Дальше вам решать.

3.2.2. Базы данных пользователей

В созданной пользователем базе данных хранится множество объектов. Главным объектом являются, разумеется, таблицы, в которые вы помещаете все данные, необходимые для решения задач конкретной предметной области. Кроме таблиц база данных хранит пользовательские типы данных, триггеры, хранимые процедуры, индексы и др. База данных содержит как сами данные, хранящиеся в таблицах, так и метаданные, описывающие эти данные. Хранение в базе данных и метаданных является важнейшим принципом, применяемым во всех базах данных. Это позволяет уменьшить зависимость программ от структуры базы данных.

Помимо пользовательских объектов в базе данных хранятся системные объекты — системные таблицы, системные представления (их очень большое количество), системные хранимые процедуры (их еще больше), системные функции (многие из них мы рассмотрим в этой книге), системные типы данных (мы подробно рассмотрим все существующие в системе типы данных), а также пользователи, роли, схемы.

Для любой базы данных используется не менее двух файлов операционной системы — файл данных (data) для хранения собственно данных и файл журнала транзакций (transaction log, иногда этот файл называют *протоколом* транзакций). Каждый из этих файлов может принадлежать только одной базе данных.

База данных может содержать не более 32767 файлов для хранения данных. Первый или единственный файл данных называется *первичным файлом*. И при вновь созданной, так сказать "пустой", базе данных в первичном файле хранятся системные данные, такие как ссылки на другие, вторичные файлы данных и на файлы журнала транзакций. Начальный размер первичного файла не может быть меньше, чем 3 Мбайта.

При желании вы можете использовать *вторичные файлы* для хранения данных. Во вторичных файлах хранятся только пользовательские данные. Файлы данных могут объединяться в файловые группы. В любой базе данных всегда присутствует первичная файловая группа PRIMARY. Если не создано никакой вторичной файловой группы, то все файлы данных принадлежат первичной группе. В некоторых случаях имеет смысл объединять отдельные файлы в файловые группы с целью повышения производительности системы. В файловые группы могут объединяться только файлы данных, но не файлы журнала транзакций.

Все файлы данных имеют страничную организацию. Размер страницы в SQL Server имеет значение 8 Кбайт и не может быть изменен.

Журнал транзакций также может быть представлен несколькими файлами. В журнале хранятся все изменения базы данных, выполненные в контексте каждой транзакции. Прежде чем записать выполненные пользователем изменения в файл данных, система вначале осуществляет необходимые записи в журнал транзакций.

Журнал используется для выполнения операций подтверждения (COMMIT) или отката (ROLLBACK) транзакций, а также для целей восстановления базы данных на любой заданный момент времени или в случае ее разрушения. Подробнее о транзакциях см. в *главе 10*. Размер файла журнала транзакций не может быть задан менее чем 512 Кбайт.

Для одного экземпляра сервера базы данных может существовать до 32767 баз данных. Каждая база данных может содержать не более 32767 файлов и не более 32767 файловых групп. Вряд ли вам когда-либо потребуется такое количество баз данных в одном экземпляре сервера и такое количество файлов в базе данных.

3.2.3. Некоторые характеристики базы данных

Каждая база данных имеет множество характеристик. Характеристики базы данных, их значения по умолчанию и средства, используемые для изменения текущих значений, описаны в *приложении 4*.

Рассмотрим некоторые из этих характеристик.

3.2.3.1. Владелец базы данных (Owner)

Владелец базы данных (owner) имеет все полномочия к базе данных. Он может изменять характеристики базы, удалять ее, вносить любые изменения в данные и метаданные. Владелцем базы становится пользователь, создавший базу данных.

Владельца пользовательской базы данных можно изменить, используя в языке Transact-SQL системную процедуру `sp_changedbowner`. Вот несколько упрощенный синтаксис обращения к этой процедуре:

```
EXECUTE sp_changedbowner '<имя нового владельца>'
```

Функция возвращает значение 0 при успешной смене владельца или 1 в случае возникновения ошибки. Например, для изменения владельца текущей базы данных можно выполнить следующее обращение к этой процедуре:

```
EXECUTE sp_changedbowner 'anotherowner'
```

Имя нового владельца уже должно быть описано в системе. Чтобы получить список существующих в системе пользователей, можно выполнить системную хранимую процедуру `sp_helplogins`:

```
EXEC sp_helplogins;
```

3.2.3.2. Порядок сортировки (collation)

Порядок сортировки (collation) для базы данных определяет допустимый набор символов в строковых типах данных CHAR, VARCHAR и правила, по которым будут при необходимости упорядочиваться эти строковые данные. Порядок сортировки определяет, будет ли сортировка происходить по внутреннему коду или в лексикографическом (алфавитном) порядке, в каком порядке будут размещаться строчные и прописные буквы, как распределяются знаки препинания, иные специальные сим-

волы и др. Если для строкового типа данных явно не указан порядок сортировки, то ему будет присвоен порядок, заданный по умолчанию для всей базы данных. Для элемента данных при его описании в базе можно указать любой допустимый порядок сортировки, отличный от порядка сортировки базы данных.

3.2.3.3. Возможность изменения данных базы данных

База данных может находиться в состоянии только для чтения (`READ_ONLY`) или доступна как для чтения, так и для внесения изменений в данные (`READ_WRITE`).

3.2.3.4. Состояние базы данных (Database State)

В каждый момент времени любая база находится в одном конкретном состоянии (state). В SQL Server существуют следующие состояния базы данных.

- ◆ **ONLINE.** База данных в доступном состоянии (или, в другой терминологии, находится в *оперативном режиме*). С ней можно выполнять любые действия по изменению данных и метаданных. В этом состоянии средствами операционной системы невозможно удалить или даже скопировать файлы базы данных на другие устройства (при запущенном на выполнение сервере базы данных).
- ◆ **OFFLINE.** База данных в недоступном состоянии (или еще говорят, что она находится в *автономном режиме*). Никакие действия с объектами базы данных в этом состоянии невозможны. Однако средствами операционной системы можно удалить файлы базы данных, чего делать все-таки не стоит, или скопировать их на другой носитель.
- ◆ **RESTORING.** База данных недоступна. В это состояние она переводится, когда выполняется восстановление файлов данных из резервной копии.
- ◆ **RECOVERING.** База данных недоступна, она находится в процессе восстановления. После завершения восстановления база автоматически будет переведена в оперативное состояние (`ONLINE`).
- ◆ **RECOVERY_PENDING.** Это состояние ожидания исправления ошибок восстановления базы данных. База данных недоступна. В процессе восстановления базы произошла ошибка, которая требует вмешательства пользователя. После исправления ошибки пользователь сам должен перевести базу в оперативное состояние.
- ◆ **SUSPECT.** База данных недоступна. Она помечена как подозрительная и может быть поврежденной. Со стороны пользователя требуются действия по устранению ошибок.
- ◆ **EMERGENCY.** База данных повреждена и находится в состоянии только для чтения (`READ_ONLY`). Такое состояние базы используется для ее диагностики и при попытках скопировать неповрежденные данные.

Существует еще множество других менее важных для обычной работы характеристик базы данных, присутствующих в указанных категориях. Некоторые из них мы рассмотрим далее в этой главе.

Здесь мы рассмотрим, какими способами можно отображать и изменять некоторые характеристики базы данных.

3.2.4. Некоторые характеристики файлов базы данных

Каждый файл базы данных (как файл данных, так и файл журнала транзакций) имеет свой набор характеристик.

3.2.4.1. Основные характеристики файлов базы данных

Каждый файл базы данных является либо файлом данных (rows data, строки данных), либо файлом журнала транзакций (log).

У каждого файла помимо имени, известного в операционной системе, существует и логическое имя (logical name), по которому к файлу можно обращаться в операторах Transact-SQL и при использовании различных компонентов SQL Server.

При создании или при изменении характеристик файла ему можно задать начальный размер (initial size). Для файла устанавливается также возможность автоматического увеличения размера и величины приращения — в процентах от начального размера или в абсолютных значениях единиц памяти (в килобайтах, мегабайтах, гигабайтах или терабайтах). Максимальный размер памяти, используемой для хранения файла, можно ограничить конкретной величиной или указать, что размер файла не ограничивается (unlimited).

3.2.4.2. Состояния файлов базы данных

Файлы данных базы данных также могут находиться в различных состояниях, причем состояние файла базы данных поддерживается независимо от состояния самой базы данных.

Файл базы данных может находиться в одном из следующих состояний.

- ◆ **ONLINE.** Файл в доступном состоянии (в *оперативном режиме*). С данными, содержащимися в этом файле, можно выполнять любые действия.
- ◆ **OFFLINE.** Файл в недоступном состоянии (в *автономном режиме*). Перевод файла в автономный и оперативный режим осуществляется пользователем. Обычно файл переводят в состояние **OFFLINE**, если он поврежден и требует восстановления. После восстановления поврежденного файла пользователь должен явно перевести его в состояние **ONLINE**.
- ◆ **RESTORING.** Файл находится в процессе восстановления. Другие действия с данными, хранящимися в этом файле, невозможны. После завершения восстановления файл автоматически переводится системой в состояние **ONLINE**.
- ◆ **RECOVERY_PENDING.** Файл автоматически переводится в это состояние, если при его восстановлении произошла ошибка. Восстановление файла было отложено. Требуется соответствующее действие от пользователя для устранения ошибки. После наведения порядка с файлом пользователь вручную переводит его в оперативное состояние.
- ◆ **SUSPECT.** В процессе оперативного восстановления файла произошла ошибка. База данных также помечается как подозрительная.
- ◆ **DEFUNCT.** Файл был удален (разумеется, когда он не был в состоянии **ONLINE**).

Если при работе с базой данных отдельные файлы являются недоступными, то все же возможны различные операции с базой данных, если для выполнения этих операций требуются только данные, содержащиеся в файлах, которые находятся в оперативном режиме. Это одно из неоспоримых преимуществ системы MS SQL Server.

ЗАМЕЧАНИЕ

Наверняка при первом знакомстве с SQL Server смысл многих вещей, таких как некоторые состояния базы данных и ее файлов, а также понимание того, что именно в различных ситуациях должен сделать пользователь, часто остается загадкой. Не расстраивайтесь. В процессе приобретения опыта использования системы все станет на свои места, и вы во всем разберетесь. И я вместе с вами, надеюсь, тоже.

3.3. Получение сведений о базах данных и их файлах в текущем экземпляре сервера

Получить сведения о базах данных (как системных, так и пользовательских) и об их файлах можно при использовании множества разнообразных средств, входящих в состав SQL Server, которыми являются:

- ◆ системные представления;
- ◆ системные хранимые процедуры;
- ◆ системные функции;
- ◆ диалоговые средства компонента Management Studio.

3.3.1. Системное представление *sys.databases*

Для того чтобы просмотреть список и характеристики всех баз данных, существующих в текущем экземпляре сервера, как пользовательских, так и системных, мы можем обратиться, пожалуй, к самому важному и информативному системному представлению просмотра каталогов *sys.databases*.

Это представление отображает значения множества характеристик баз данных текущего экземпляра сервера. Представление возвращает одну строку для каждой базы данных. Рассмотрим только некоторые из столбцов, получаемых из этого представления, которые нам будут в первую очередь интересны.

- ◆ *name* — содержит логическое имя базы данных.
- ◆ *database_id* — идентификатор базы данных (целое число), автоматически присваиваемый базе данных системой при ее создании.
- ◆ *create_date* — дата и время создания базы данных. Время указывается с точностью до миллисекунд.
- ◆ *collation_name* — имя порядка сортировки по умолчанию для базы данных.
- ◆ *is_read_only* — определяет, является ли база данных базой только для чтения:
 - 0 — база данных находится в режиме только для чтения (*READ_ONLY*);
 - 1 — база данных в режиме чтения и записи (*READ_WRITE*).

◆ `state` — состояние базы данных:

- 0 — ONLINE;
- 1 — RESTORING;
- 2 — RECOVERING;
- 3 — RECOVERY_PENDING;
- 4 — SUSPECT;
- 5 — EMERGENCY;
- 6 — OFFLINE.

◆ `state_desc` — текстовое описание состояния базы данных: ONLINE, RESTORING, RECOVERING, RECOVERY_PENDING, SUSPECT, EMERGENCY, OFFLINE. Эта характеристика является, разумеется, производной от состояния базы данных (`state`).

3.3.2. Системное представление *sys.master_files*

Другое полезное системное представление — `sys.master_files`. Оно позволяет получить детальный список всех баз данных и файлов, входящих в состав каждой базы данных, а также многие интересные характеристики этих файлов.

Наиболее важными для нас столбцами этого представления будут следующие.

◆ `database_id` — идентификатор базы данных. Имеет то же значение, что и в представлении `sys.databases`.

◆ `file_id` — идентификатор файла (тоже целое число). Первичный файл имеет идентификатор 1.

◆ `type` — задает тип файла:

- 0 — файл данных (ROWS);
- 1 — журнал транзакций (LOG);
- 2 — файловый поток (FILESTREAM).

◆ `type_desc` — текстовое описание типа файла из столбца `type`: ROWS — файл данных, LOG — файл журнала транзакций, FILESTREAM — файловый поток.

◆ `data_space_id` — идентификатор пространства данных (файловой группы), которому принадлежит файл данных. Для всех файлов журнала транзакций идентификатор имеет значение 0.

◆ `name` — логическое имя файла, заданное в операторе CREATE DATABASE или присвоенное системой по умолчанию, если пользователь не указал логическое имя.

◆ `physical_name` — путь к файлу, включая имя дискового устройства, и имя файла в операционной системе.

◆ `state` — состояние файла:

- 0 — ONLINE;
- 1 — RESTORING;

- 2 — RECOVERING;
 - 3 — RECOVERY_PENDING;
 - 4 — SUSPECT;
 - 6 — OFFLINE;
 - 7 — DEFUNCT.
- ◆ `state_desc` — текстовое описание состояния файла, производное от состояния файла (`state`): ONLINE, RESTORING, RECOVERING, RECOVERY_PENDING, SUSPECT, OFFLINE, DEFUNCT.
- ◆ `is_read_only` — определяет, является ли файл файлом только для чтения:
- 1 — файл READ_ONLY, только для чтения;
 - 0 — файл READ_WRITE, возможны операции чтения, добавления, изменения и удаления данных в этом файле.
- ◆ `size` — размер файла в страницах. Напомню, что страница в файле данных имеет размер 8 Кбайт или, иными словами, 8192 байта.
- ◆ `max_size` — указывает три возможных варианта: (1) возможность увеличения размера файла, (2) максимальный размер файла в страницах или (3) неограниченность размера файла. Может принимать следующие значения:
- 0 — увеличение размера файла недопустимо;
 - -1 — файл растет неограниченно, пока он не исчерпает объема всего дискового пространства или пока не достигнет допустимого предела (2 Тбайт для журнала транзакций или 16 Тбайт для файла данных);
 - положительное число — указывает максимальный размер файла в страницах. Число 268435456 может быть указано только для файла журнала транзакций. Означает, что файл может расти до максимального размера в 2 Тбайт.
- ◆ `is_percent_growth` — указывает, задается ли увеличение размера файла в процентах или в страницах:
- 0 — увеличение размера указано в страницах;
 - 1 — увеличение размера файла указано в процентах от начального размера.
- ◆ `growth` — указывает, будет ли увеличиваться размер файла:
- 0 — файл не будет увеличиваться в размерах. Указанный размер не может изменяться;
 - 1 и более — размер файла будет при необходимости увеличиваться автоматически в соответствии с заданными параметрами при создании базы данных.

3.3.3. Системное представление `sys.database_files`

Представление `sys.database_files` позволяет получить список файлов и их характеристик только одной текущей базы данных, указанной в операторе USE.

Вот некоторые характеристики, которые можно получить при вызове этого представления. Они в точности дублируют значения столбцов представления `sys.master_files`. Вкратце повторим эти значения.

- ◆ `file_id` — идентификатор файла.
- ◆ `type` — тип файла: 0 — файл данных (`ROWS`), 1 — журнал транзакций (`LOG`), 2 — файловый поток (`FILESTREAM`).
- ◆ `type_desc` — текстовое описание типа файла: `ROWS` — файл данных, `LOG` — файл журнала транзакций, `FILESTREAM` — файловый поток.
- ◆ `data_space_id` — идентификатор файловой группы, которой принадлежит файл.
- ◆ `name` — логическое имя файла, явно заданное в операторе `CREATE DATABASE` или присвоенное системой по умолчанию.
- ◆ `physical_name` — путь к файлу, включая имя дискового устройства, каталоги и имя файла в операционной системе.
- ◆ `state` — состояние файла:
 - 0 — `ONLINE`;
 - 1 — `RESTORING`;
 - 2 — `RECOVERING`;
 - 3 — `RECOVERY_PENDING`;
 - 4 — `SUSPECT`;
 - 6 — `OFFLINE`.
 - 7 — `DEFUNCT`.
- ◆ `state_desc` — текстовое описание состояния файла, производное от состояния файла (`state`): `ONLINE`, `RESTORING`, `RECOVERING`, `RECOVERY_PENDING`, `SUSPECT`, `OFFLINE`, `DEFUNCT`.
- ◆ `is_read_only` — определяет, является ли файл файлом только для чтения:
 - 1 — файл `READ_ONLY`, только для чтения;
 - 0 — файл `READ_WRITE`, возможны операции чтения, удаления и обновления данных в этом файле.
- ◆ `size` — размер файла в страницах.
- ◆ `max_size` — указывает возможность увеличения размера файла, максимальный размер файла в страницах или неограниченность размера файла. Может принимать следующие значения:
 - 0 — увеличение размера файла недопустимо;
 - -1 — файл растет неограниченно, пока не исчерпает всего дискового пространства или не достигнет допустимого предела (2 Тбайта для журнала транзакций или 16 Тбайт для файла данных);
 - положительное число — указывает максимальный размер файла в страницах.

- ◆ `is_percent_growth` — указывает, задается ли увеличение размера файла в процентах или в страницах:
 - 0 — увеличение размера указано в страницах;
 - 1 — увеличение размера файла указано в процентах от начального размера.
- ◆ `growth` — указывает, будет ли увеличиваться размер файла:
 - 0 — файл не будет увеличиваться в размерах. Указанный размер не может изменяться;
 - 1 и более — размер файла будет при необходимости увеличиваться автоматически в соответствии с заданными параметрами при создании базы данных.

3.3.4. Системное представление *sys.filegroups*

Системное представление `sys.filegroups` позволяет получить некоторые данные о файловых группах текущей базы данных, указанной в операторе `USE`. Вот некоторые столбцы этого представления, которые могут быть нам интересны.

- ◆ `name` — название файловой группы. Первой всегда будет первичная файловая группа `PRIMARY`.
- ◆ `type` — тип. Для файловых групп имеет значение `FG`.
- ◆ `type_desc` — текстовое описание типа. Для файловой группы имеет значение `ROWS_FILEGROUP`.
- ◆ `is_read_only` — указывает, является ли файловая группа группой только для чтения:
 - 0 — файловая группа доступна для чтения и записи, т. е. все файлы, входящие в состав этой файловой группы, доступны для чтения и записи;
 - 1 — файловая группа только для чтения.

* * *

Это, пожалуй, наиболее важные и полезные представления, которые вы будете использовать в вашей повседневной жизни. Давайте теперь очень кратко рассмотрим и некоторые другие средства, которые вам могут оказаться полезными при работе с базами данных в SQL Server.

3.3.5. Другие средства получения сведений об объектах базы данных

Мы рассмотрели некоторые средства, позволяющие получить детальные сведения о характеристиках существующих баз данных, их файлах и файловых группах. Имеет смысл сказать несколько слов и о других средствах, которые мы с вами будем использовать в ближайшем будущем.

3.3.5.1. Системные представления

Представления для получения сведений об объектах базы данных.

- ◆ `sys.schemas` — возвращает сведения о схемах базы данных. Каждая база данных может содержать более двух миллиардов схем. Надо сказать, что "схема" (schema) в SQL Server, это совсем не то, что схема в некоторых других системах управления базами данных. О схемах мы поговорим в *разд. 3.8*.
- ◆ `sys.database_permissions` — возвращает сведения о полномочиях в базе данных. Вопросы безопасности в SQL Server решаются довольно жестко и очень разумно. Специалистам по системам безопасности, в том числе и в базах данных, следует этим делам уделить достаточно серьезное внимание.
- ◆ `sys.database_principals` — возвращает сведения о принципах (владельцах или, иными словами, участниках доступа к базам данных и их объектам) в базе данных. Это опять же связано с вопросами безопасности в базах данных.
- ◆ `sys.database_role_members` — возвращает сведения о членах (участниках) ролей в базе данных. И роли базы данных при правильном их использовании могут быть хорошим средством для повышения безопасности баз данных.

Другими системными представлениями просмотра каталогов для объектов, скажем так, "детального уровня", являются следующие далее представления.

- ◆ `sys.tables` — возвращает сведения о таблицах базы данных. Здесь даются сведения обо всех таблицах текущей базы данных.
- ◆ `sys.views` — возвращает сведения о представлениях в базе данных. Каждая база данных может содержать множество различных представлений. Представления позволяют упростить задачу пользователя по получению данных из одной или нескольких базовых таблиц. Представления позволяют "скрыть" от не очень профессионального (или слишком ленивого) пользователя все сложности, связанные с составлением запроса к данным базы данных для получения необходимых результатов.
- ◆ `sys.indexes` — возвращает сведения об индексах базы данных. *Индекс* — это замечательный объект реляционной базы данных, который в одно и то же время может резко ухудшить работу с данными в базе данных, а может при правильном проектировании и сильно повысить производительность при выборке и упорядочении данных базы. Однако добавление к таблицам новых индексов никак не может улучшить временные характеристики при выполнении операций добавления или изменения данных, если в процесс изменения включены столбцы, входящие в состав индекса.
- ◆ `sys.events` — возвращает сведения о событиях базы данных. *Событие* — это очень интересное и полезное средство при работе с базами данных. События позволяют синхронизировать работу программ, одновременно использующих одну и ту же базу, и иногда избежать некоторых ошибок в работе пользователей с базой данных.

- ◆ `sys.types` — возвращает сведения о системных и пользовательских типах данных.
- ◆ `sys.columns` — возвращает сведения о столбцах таблиц и представлений.

Для обращения к системным представлениям, как и к другим представлениям в базе данных, мы используем оператор `SELECT`, синтаксис которого и варианты применения будем подробно рассматривать в этой главе и в последующих главах.

3.3.5.2. Системные хранимые процедуры

Для обращения к хранимым процедурам, как системным, так и к пользовательским, используется оператор `EXECUTE`. Примеры таких обращений мы вскоре рассмотрим.

Часто используются следующие хранимые процедуры.

- ◆ `sp_databases` — предоставляет список баз данных.
- ◆ `sp_stored_procedures` — возвращает список хранимых процедур.
- ◆ `sp_help` — возвращает список различных объектов базы данных, типов данных, определенных пользователем или поддерживаемых системой SQL Server.
- ◆ `sp_helplogins` — возвращает список регистрационных имен пользователей (login).
- ◆ `sp_helptext` — дает возможность получить на языке Transact-SQL тексты, описывающие системные хранимые процедуры, триггеры, вычисляемые столбцы, ограничения `CHECK` для столбцов таблиц.
- ◆ `sp_changedbowner` — позволяет изменить владельца базы данных.
- ◆ `sp_configure` — позволяет изменить некоторые режимы системы.

3.3.5.3. Системные функции

Помимо системных представлений просмотра каталогов и системных хранимых процедур в SQL Server присутствуют системные функции.

Чтобы просмотреть логические имена файлов любой базы данных, системной или пользовательской, можно вызвать системную функцию `FILE_NAME()`.

Для получения идентификатора базы данных, который присваивается ей при ее создании, используется функция `DB_ID()`. Очень удобная и довольно часто применяемая функция. Ее регулярно будем использовать как в этой главе, так и в реальной жизни.

Имя базы данных можно получить при вызове функции `DB_NAME()`, которой передается в качестве параметра идентификатор базы данных.

Функция `FILE_ID()` для текущей базы данных возвращает идентификатор файла базы данных по указанному логическому имени этого файла.

Функция `FILEGROUP_ID()` возвращает идентификатор файловой группы, заданной ее именем.

Функция `FILEGROUP_NAME()`, наоборот, по идентификатору файловой группы возвращает ее имя.

* * *

В нужное время мы обратимся к этим системным представлениям, хранимым процедурам и функциям и станем использовать их по прямому назначению для получения необходимых сведений об объектах и характеристиках базы данных.

3.4. Создание и удаление базы данных

В этом разделе мы рассмотрим всевозможные способы создания пользовательских баз данных при применении как оператора Transact-SQL `CREATE DATABASE`, так и диалоговых средств Management Studio. Кроме создания баз данных мы научимся их удалять при помощи простого оператора `DROP DATABASE`, а также при использовании диалоговых средств Management Studio.

Для отображения характеристик и состояния существующей базы данных средствами Transact-SQL можно использовать описанные системные представления и диалоговые средства, существующие в Management Studio. Мы рассмотрим все подходящие варианты.

3.4.1. Использование операторов Transact-SQL для создания, отображения и удаления баз данных

3.4.1.1. Оператор создания базы данных

Для создания новой базы данных используется оператор `CREATE DATABASE`. Его синтаксис (только для целей создания новой базы данных) показан в листинге 3.1 и на графе 3.1. Детализация синтаксических конструкций представлена в графах 3.2—3.7. Основные синтаксические конструкции, как мы ранее договорились, будем представлять и в нотациях Бэкуса — Наура, и при помощи R-графов.

**Листинг 3.1. Синтаксис оператора `CREATE DATABASE`.
Вариант создания новой базы данных**

```
CREATE DATABASE <логическое имя базы данных>
[ CONTAINMENT = { NONE | PARTIAL } ]
[ <предложение ON> [ <предложение LOG ON> ] ]
[ COLLATE <порядок сортировки> ]
[ WITH <опция> [, <опция>]... ];

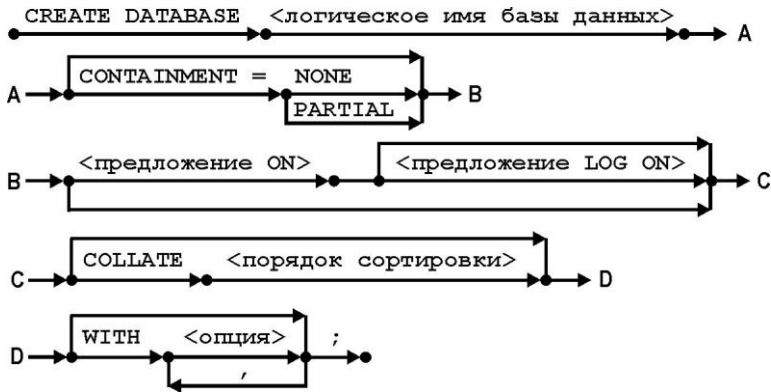
<предложение ON> ::=
ON [ PRIMARY ] <спецификация файла> [, <спецификация файла>]...
    [, <файловая группа> [, <файловая группа>] ...]

<предложение LOG ON> ::=
LOG ON <спецификация файла> [, <спецификация файла> ]...
```

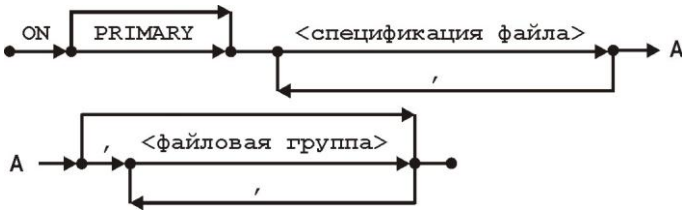
```
<опция> ::=
{ FILESTREAM (<опция файлового потока> [, <опция файлового потока> ]...)
| DEFAULT_FULLTEXT_LANGUAGE =
  { <код языка> | <название языка> | <псевдоним языка> }
| DEFAULT_LANGUAGE =
  { <код языка> | <название языка> | <псевдоним языка> }
| NESTED_TRIGGERS = { OFF | ON }
| TRANSFORM_NOISE_WORDS = { OFF | ON }
| TWO_DIGIT_YEAR_CUTOFF = <год между 1753 и 9999>
| DB_CHAINING { OFF | ON }
| TRUSTWORTHY { OFF | ON }
}
```



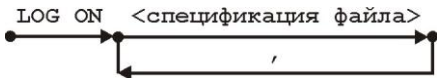
```
<опция файлового потока> ::=
{ NON_TRANSACTED_ACCESS = { OFF | READ_ONLY | FULL }
| DIRECTORY_NAME = '<имя каталога>'
}
```



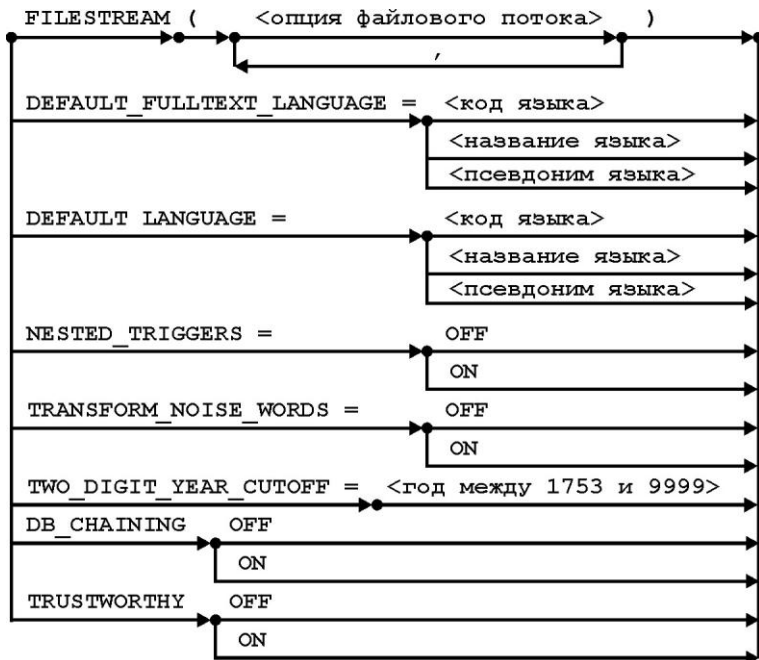
Граф 3.1. Синтаксис оператора CREATE DATABASE (создание базы данных)



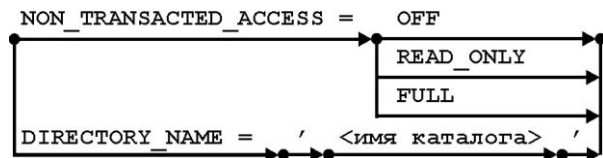
Граф 3.2. Синтаксис предложения ON



Граф 3.3. Синтаксис предложения LOG ON



Граф 3.4. Нетерминальный символ "опция"



Граф 3.5. Нетерминальный символ "опция файлового потока"

ЗАМЕЧАНИЕ

Оператор `CREATE DATABASE` с несколько измененным синтаксисом может также использоваться и для некоторых иных целей: присоединения где-то кем-то когда-то созданной базы данных, возможно, на другом компьютере к списку баз данных текущего экземпляра сервера, а также для создания так называемого мгновенного снимка базы данных (`SNAPSHOT`). Все это мы рассмотрим чуть позже в данной главе.

В результате успешного создания новой базы данных будут созданы все специфицированные в операторе или сформированные по умолчанию с соответствующими характеристиками файлы базы данных. Самой базе данных будут присвоены заданные явно или по умолчанию значения ее характеристик. Файлы данных могут быть объединены в файловые группы.

Как видно из синтаксиса, при создании базы данных обязательно нужно задать только лишь логическое имя базы данных, которое будет известно в текущем экземпляре сервера. Все остальные конструкции являются необязательными. В этом случае самой базе данных и файлам создаваемой базы данных будут присвоены

значения всех характеристик по умолчанию, которые мы с вами и рассмотрим в ближайшее время.

ЗАМЕЧАНИЕ ПО СИНТАКСИСУ

Во множестве предыдущих версий документации по SQL Server сообщалось, что начиная с данной версии все операторы Transact-SQL должны завершаться символом точка с запятой (;). Это не очень соответствовало реальной действительности. И тогда операторы можно было не завершать этим символом. Лучшим вариантом будет, если вы *всегда* станете заканчивать любой оператор Transact-SQL символом точка с запятой.

Логическое имя базы данных

Логическое имя базы данных является обязательным параметром. Оно идентифицирует создаваемую базу данных. Имя должно соответствовать правилам задания идентификаторов, обычных или с разделителями (см. главу 2). В обычном идентификаторе здесь также можно использовать и буквы кириллицы, если при установке сервера вы задали порядок сортировки Cyrillic_General_CI_AS. В идентификаторах с разделителями, как вы помните, можно использовать *любые* символы. Такой идентификатор только нужно заключить в квадратные скобки. Имя не должно содержать более 128 символов. Имя должно быть уникальным среди имен всех баз данных текущего экземпляра SQL Server. По этому имени вы обращаетесь к конкретной базе данных экземпляра сервера во всех операторах Transact-SQL, где требуется указать базу данных. Это единственный обязательный параметр в данном операторе.

Предложение *CONTAINMENT*

```
[ CONTAINMENT = { NONE | PARTIAL } ]
```

Необязательное предложение *CONTAINMENT* определяет степень независимости базы данных от характеристик экземпляра сервера базы данных, в котором создается база данных. Если указано *NONE* (значение по умолчанию), то создается обычная (неавтономная) база данных.

Если же указано значение *PARTIAL*, то создается так называемая "partially contained" база данных, т. е. частично автономная. Пользователь может подключаться к такой базе данных, используя отдельные средства аутентификации, не связанные с уровнем экземпляра сервера. Такую базу данных проще перенести в другой экземпляр сервера базы данных, на другой компьютер. Кроме того, в таких базах данных можно избежать неприятностей, которые иногда возникают при создании временных таблиц, где присутствуют строковые столбцы с порядком сортировки, отличным от принятого по умолчанию.

Подобные базы данных мы позже рассмотрим в примерах.

Предложение *ON*

В операторе создания базы данных может задаваться первичный файл данных в предложении *ON*. Файл, описанный первым в предложении, а также перечисленные

вслед за ним файлы, если они указаны, помещаются в файловую группу `PRIMARY`. О вторичных файловых группах см. чуть дальше.

Все описания файлов заключаются в круглые скобки (см. далее синтаксис спецификации файла в разд. *"Спецификация файла"*) и отделяются друг от друга запятыми.

База данных помимо обязательной первичной файловой группы `PRIMARY` может содержать большое количество других, вторичных, файловых групп. Вторичные файловые группы вместе с принадлежащими им файлами данных описываются после первичной файловой группы.

Предложение **LOG ON**

Необязательное предложение `LOG ON` описывает файл (файлы) журнала транзакций. В базе данных может существовать большое количество файлов журналов транзакций. Предложение `LOG ON` можно опустить. В этом случае файлу журнала транзакций присваиваются значения по умолчанию.

Задание нескольких файлов журналов транзакций имеет смысл только в том случае, если для одного файла не хватает места на внешнем носителе. Тогда администратор БД создает файл (файлы) на другом носителе (других носителях).

ЗАМЕЧАНИЕ

Как видно из приведенного синтаксиса оператора `CREATE DATABASE`, при создании базы данных можно вообще не задавать никаких файлов данных и файлов журнала транзакций (эти конструкции заключены в описании синтаксиса в обрамляющие квадратные скобки). Если приглядеться внимательно к этому описанию, то можно увидеть, что допустим вариант задания файла (файлов) данных при отсутствии задания файлов журнала транзакций. Однако указать файл (файлы) журнала транзакций при отсутствии описания файла данных нельзя. В таком случае вы получите ошибку системы.

Если при создании новой базы данных вы не укажете файл данных или файл журнала транзакций, то система всем характеристикам этих файлов присвоит значения по умолчанию (см. далее).

Предложение **COLLATE**

Необязательное предложение `COLLATE` позволяет задать для создаваемой базы данных порядок сортировки, отличный от того, который был установлен при установке системы для экземпляра сервера. Если предложение не указано, то база данных будет иметь порядок сортировки по умолчанию, заданный при установке системы. Мы с вами при установке SQL Server установили `Cyrillic_General_CI_AS`.

Предложение **WITH**

Необязательное предложение `WITH` позволяет описать некоторые дополнительные характеристики создаваемой базы данных.

```
FILESTREAM (<опция файлового потока> [, <опция файлового потока> ]...)  
<опция файлового потока> ::=  
{ NON_TRANSACTED_ACCESS = { OFF | READ_ONLY | FULL }  
  | DIRECTORY_NAME = '<имя каталога>' }
```


После ключевого слова `FILESTREAM` в скобках перечисляются опции файлового потока.

Опция `NON_TRANSACTIONED_ACCESS` определяет возможность доступа к данным файлового потока вне контекста транзакций. Транзакции мы рассмотрим в *главе 10*. Значениями опции являются:

- ◆ `OFF` — доступ к данным вне транзакции недопустим;
- ◆ `READ_ONLY` — к данным файлового потока возможен доступ вне транзакций только для операций чтения;
- ◆ `FULL` — допустимы все операции к данным файлового потока вне транзакций.

Опция `DIRECTORY_NAME` задает имя каталога. Это имя должно быть уникальным среди имен каталогов, заданных параметром `DIRECTORY_NAME` текущего экземпляра сервера. Каталог с этим именем будет создан внутри сетевого каталога экземпляра сервера базы данных. Используется для файловых таблиц (`FileTable`), которые мы рассмотрим в *главе 5*.

Опция `DEFAULT_FULLTEXT_LANGUAGE` (допустимо только для автономной базы данных) задает язык базы данных по умолчанию для полнотекстового поиска в индексированных столбцах. Язык может задаваться в виде кода языка, его названия или псевдонима. Список допустимых языков см. в *приложении 5*.

Опция `DEFAULT_LANGUAGE` (допустимо только для автономной базы данных) задает язык по умолчанию для вновь создаваемых регистрационных имен пользователей. Может задаваться в виде кода языка, его названия или псевдонима.

Опция `NESTED_TRIGGERS` (допустимо только для автономной базы данных) задает возможность использования вложенных триггеров `AFTER`. Если указано `OFF`, вложенные триггеры недопустимы. При задании `ON` может существовать до 32 уровней триггеров. То есть триггер может вызывать (разумеется, неявно) другой триггер, который, в свою очередь, инициирует обращение к триггеру следующего уровня. И так до 32 уровней.

Опция `TRANSFORM_NOISE_WORDS` (допустимо только для автономной базы данных) задает поведение сервера базы данных в некоторых ситуациях полнотекстового поиска в базе данных. Существует понятие *noise word* или *stop word*. Это слова, которые слишком часто присутствуют в текстах и не имеют особого смысла при выполнении поисковых действий. Чаще всего это предлоги, для некоторых языков артикли, а также множество других часто встречающихся в языке слов.

Значение `OFF` (по умолчанию) приводит к тому, что если в запросе встречаются такие слова и запрос возвращает нулевое количество строк, то просто выдается предупреждающее сообщение.

Если указано `ON`, то система выполнит преобразование запроса, удалив из него соответствующие слова.

Опция `TWO_DIGIT_YEAR_CUTOFF` (допустимо только для автономной базы данных) задает значение года в диапазоне между 1753 и 9999. По умолчанию принимается 2049. Это число используется для интерпретации года, заданного двумя символами.

Если двухсимвольный год меньше или равен последним двум цифрам указанного четырехсимвольного значения, то этот год будет интерпретироваться как год того же столетия. Если больше — то будет использовано столетие, следующее за указанным в операторе столетием. Хорошей практикой является указание во *всех* датах четырехсимвольного значения года.

Опция `DB_CHAINING` определяет, может ли создаваемая база данных использоваться в цепочках связей между несколькими базами данных. `OFF` (по умолчанию) запрещает такое использование, `ON` — разрешает.

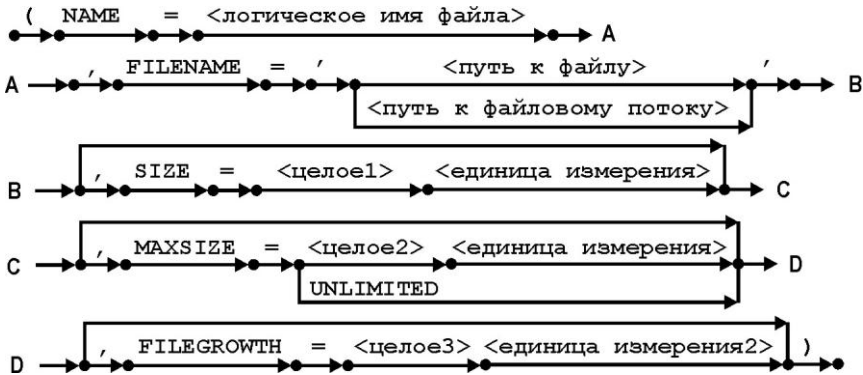
Опция `TRUSTWORTHY` задает, могут ли программные компоненты базы данных (хранимые процедуры, представления, созданные пользователем функции) обращаться к ресурсам вне базы данных. `OFF` (по умолчанию) запрещает обращение к внешним ресурсам, `ON` — разрешает.

Спецификация файла

Синтаксис спецификации файла, одинаковый как для файлов данных, так и для журналов транзакций, показан в листинге 3.2 и в соответствующем R-графе (граф 3.6).

Листинг 3.2. Синтаксис спецификации файла

```
<спецификация файла> ::=
( NAME = <логическое имя файла>,
  FILENAME = { '<путь к файлу>' | '<путь к файловому потоку>' }
  [, SIZE = <целое1> [ KB | MB | GB | TB ] ]
  [, MAXSIZE = { <целое2> [ KB | MB | GB | TB ] | UNLIMITED } ]
  [, FILEGROWTH = <целое3> [ KB | MB | GB | TB | % ] ]
)
```



Граф 3.6. Синтаксис спецификации файла

В спецификации файла обязательными являются только предложения `NAME` (логическое имя файла) и `FILENAME` (имя файла в операционной системе). В случае отсутствия любого из предложений `SIZE`, `MAXSIZE` или `FILEGROWTH` соответствующим харак-

теристикам будут присвоены значения по умолчанию, которые мы рассмотрим далее. Эти значения по умолчанию различны для файлов данных и для файлов журнала транзакций.

В R-графе указаны элементы "единица измерения" и "единица измерения2". Вторым элементом отличается от первого тем, что в его списке присутствует и знак процента.

Предложение **NAME**

NAME = <логическое имя файла>

Предложение NAME задает логическое имя файла. Это имя может использоваться при различных ссылках на данный файл. Оно должно быть уникальным только в этой базе данных.

Если ни один *файл данных* явно при создании базы данных не описывается (отсутствует предложение ON), то логическому имени единственного файла данных присваивается имя самой базы данных. Например, если создаваемая база данных имеет имя Strange, то и логическое имя файла данных по умолчанию будет Strange.

Если не задается ни одного *файла журнала транзакций* (не указано предложение LOG ON), то логическому имени единственного файла журнала транзакций присваивается имя, состоящее из имени базы данных, к которому добавляется суффикс _log. Для той же базы данных Strange при отсутствии явного задания файла журнала транзакций логическое имя этого файла будет Strange_log.

Предложение **FILENAME**

FILENAME = { ' <путь к файлу>' | ' <путь к файловому потоку>' }

Предложение FILENAME определяет полный путь к файлу (включая имя внешнего носителя), а также имя самого создаваемого файла. Путь (все каталоги в пути) должен существовать на указанном внешнем носителе, а сам файл должен отсутствовать. Для первичного файла данных принято использовать расширение mdf, для вторичных файлов данных — расширение ndf, а для журнала транзакций — ldf. Такие значения расширений не являются обязательными, однако следование этому правилу опять же повышает читаемость скриптов и удобство в работе с системой. Файл (файлы) журнала транзакций следует размещать на устройствах, отличных от тех, на которых размещаются файлы данных. В случае любых сбоев дисковых носителей это позволит выполнить восстановление базы данных при наличии резервной копии. Кроме того, размещение файлов журнала транзакций на *физических* носителях, отличных от физических носителей для хранения файлов данных, повышает производительность системы, уменьшая количество перемещений головок диска при операциях чтения-записи данных. Однако в случае установок значений по умолчанию эта рекомендация не выполняется. Для достаточно простых баз данных или в целях демонстрационного или исследовательского характера все файлы можно смело размещать на одном и том же носителе и в одном и том же каталоге, что мы в большинстве случаев и делаем в рамках данной книги.

Если файл данных и файл журнала транзакций в операторе `CREATE DATABASE` явно не описываются, то для их размещения выбираются пути по умолчанию, заданные при установке системы. В нашем случае это будут пути

`C:\Program Files\Microsoft SQL Server\MSSQL11.MSSQLSERVER\MSSQL\DATA`

ЗАМЕЧАНИЕ

Пути по умолчанию для файла данных и файла журнала транзакций можно изменить в Management Studio. Для этого в **Object Explorer** нужно щелкнуть правой кнопкой мыши по имени сервера базы данных и в контекстном меню выбрать **Properties**. В появившемся окне свойств сервера нужно выбрать вкладку **Database Settings** и изменить пути в полях **Data** и **Log** для файлов данных и журналов транзакций соответственно.

Именам файлов присваивается логическое имя файла с расширением `mdf` (файл данных) или `ldf` (журнал транзакций). Например, в приведенном только что примере с созданием базы данных **Strange** файл данных получит имя **Strange.mdf**, а файл журнала транзакций — **Strange.ldf**. Понятно, что по умолчанию всегда будет создаваться лишь один файл данных и только один файл журнала транзакций в одном и том же каталоге на внешнем носителе.

Обратите внимание, что имя файла задается одной строковой константой. Здесь по не совсем понятной для меня причине недопустимо использование каких-либо выражений, внутренних переменных, операций. В частности, нельзя даже использовать простую операцию конкатенации строк. Первоначально складывается впечатление, что в пакете на создание базы данных динамически сформировать путь к файлу и имя файла вообще невозможно. Однако это не так. В Transact-SQL можно использовать очень полезный во многих случаях оператор `EXECUTE`, который позволяет динамически создавать в том числе и оператор `CREATE DATABASE`. Использование локальных переменных в таких ситуациях при создании базы данных см. далее в примерах. Кроме того, в утилите командной строки `sqlcmd` существует возможность вызывать на выполнение скрипт, содержащий параметры, значения которым подставляются при вызове этой утилиты. Такой пример мы также вскоре рассмотрим.

В случае, когда спецификация файла задает файл файлового потока (`filestream`), путь к файлу указывает только имена каталогов. При этом имена всех каталогов, кроме самого нижнего уровня, должны существовать в базе данных. Последний в пути каталог должен отсутствовать. Он будет создан системой.

Предложение **SIZE**

[**SIZE** = <целое1> [**KB** | **MB** | **GB** | **TB**]]

Необязательное предложение **SIZE** задает начальный размер файла. <Целое1> в предложении является целым числом, определяющим размер в указанных следом за ним единицах — в килобайтах (**KB**), мегабайтах (**MB**), гигабайтах (**GB**) или в терабайтах (**TB**). Если единица измерения не указана, то предполагаются мегабайты; обратите внимание, что в описании синтаксиса **MB** подчеркнуто, что, как мы помним, используется для указания значения по умолчанию. Ненавязчиво сообщу, что при создании реальных систем я *всегда* указываю единицу измерения.

Размер любого файла не может быть меньше, чем 512 Кбайт, а первичный файл данных должен иметь размер не менее 3 Мбайт. `<целое1>` имеет целочисленный тип данных `INTEGER`, его значение не может превышать 2147483647. Здесь речь идет только о числовом значении самого параметра, а не о размере файла. Для задания больших размеров следует указывать соответствующие единицы измерения.

Вы также не сможете задать начальный размер файла, который превышает объем свободного места на выбранном носителе вашего компьютера. При создании базы данных и при размещении ее файлов система выполнит и такую проверку.

Если предложение `SIZE` не задано, то начальному размеру файла присваивается значение по умолчанию, определенное в системной базе данных `model`. В моей версии установленной системы файл данных по умолчанию получает начальный размер 3 Мбайта, а журнал транзакций — 1 Мбайт. Вряд ли эти значения изменятся при переходе к другим версиям системы.

Предложение **MAXSIZE**

```
[ MAXSIZE = { <целое2> [ KB | MB | GB | TB ] | UNLIMITED } ]
```

Необязательное предложение `MAXSIZE` задает максимальный размер, который может получить файл при увеличении количества данных, помещаемых в файл, или указывает, что размер файла не ограничен (параметр `UNLIMITED`). В последнем случае файл будет увеличиваться в размерах на величину, указанную в ключевом слове `FILEGROWTH`, пока не будет исчерпано все свободное пространство носителя. На самом деле это не совсем так. "Неограниченность" размера означает лишь, что файл данных не может превышать 16 терабайт (для редакции Express используется другое ограничение — *вся* база данных по размеру не может превышать 10 Гбайт), а файл журнала транзакций — 2 Тбайта. Как и в случае задания начального размера файла, в этом предложении максимальный размер может задаваться в килобайтах, мегабайтах, гигабайтах и терабайтах (параметры `KB`, `MB`, `GB` и `TB`, соответственно). Значением единицы измерения по умолчанию также является мегабайт.

Если это предложение не указано, то для файла данных задается неограниченный (`unlimited`) размер.

Предложение **FILEGROWTH**

```
[ FILEGROWTH = <целое3> [ KB | MB | GB | TB | % ] ]
```

Необязательное предложение `FILEGROWTH` позволяет задать значение величины приращения размера файла. Параметр `<целое3>` задает увеличение размера в килобайтах, мегабайтах, гигабайтах, в терабайтах или в процентах от начального размера файла, как указано в этом предложении (`KB`, `MB`, `GB`, `TB`, `%`). Если единица измерения приращения не указана, то принимается мегабайт. Если это предложение вообще не указано, то для файла данных приращение задается в 1 Мбайт, а для журнала транзакций устанавливается приращение в 10% от начального размера файла.

Как уже говорилось, размер страницы файла данных в базе имеет фиксированное значение 8 Кбайт (8192 байта) и не может быть изменен ни при создании базы данных, ни при ее изменении.

Файловая группа

Синтаксис описания файловой группы представлен в листинге 3.3 и в соответствующем R-графе (граф 3.7).

Листинг 3.3. Синтаксис описания файловой группы

```
<файловая группа> ::=
FILEGROUP <имя файловой группы> [ CONTAINS FILESTREAM ] [ DEFAULT ]
<спецификация файла> [ , <спецификация файла> ] ...
```



Граф 3.7. Синтаксис описания файловой группы

Имя файловой группы должно быть уникальным среди имен файловых групп этой базы данных.

Необязательное предложение `CONTAINS FILESTREAM` означает, что данная файловая группа предназначена только для хранения в файловой системе столбцов указанной таблицы с типом данных `VARBINARY (MAX)`. Примеры использования файловых потоков мы рассмотрим в главе 5.

Ключевое слово `DEFAULT` указывает, что файловая группа является файловой группой по умолчанию в этой базе данных.

Файловой группе должно предшествовать, по меньшей мере, одно описание файла данных первичной группы. В состав файловой группы должен входить хотя бы один файл данных.

ЗАМЕЧАНИЕ ПО СИНТАКСИСУ ОПЕРАТОРА `CREATE DATABASE`

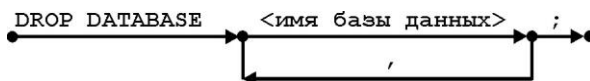
Кажется немного странным, что в языке Transact-SQL синтаксис оператора `CREATE DATABASE` для целей создания новой базы данных (другие варианты использования этого оператора мы рассмотрим ближе к концу главы) не позволяет явно установить начальные значения тому множеству характеристик базы данных, ее файловых групп и файлов, которые существуют в диалоговых средствах SQL Server. Для изменения значений характеристик по умолчанию используется оператор `ALTER DATABASE`. Его мы очень скоро будем рассматривать довольно подробно. В диалоговых средствах Management Studio и в случае первоначального создания базы данных можно задавать значения для большинства характеристик. Полагаю, что такую возможность было бы полезно внести и в оператор `CREATE DATABASE`.

3.4.1.2. Оператор удаления базы данных

Для удаления созданной пользователем и существующей в текущем экземпляре сервера базы данных используется оператор `DROP DATABASE`. Его синтаксис показан в листинге 3.4 и соответствующем R-графе (граф 3.8).

Листинг 3.4. Синтаксис оператора DROP DATABASE

```
DROP DATABASE <имя базы данных> [, <имя базы данных>]... ;
```

**Граф 3.8. Синтаксис оператора DROP DATABASE**

Оператор `DROP DATABASE` позволяет удалить одну или более указанных пользовательских баз данных или мгновенных снимков базы данных (см. *разд. 3.7*). Нельзя удалить системную базу данных. Нельзя также удалить базу данных, которая используется в настоящий момент, т. е. ту базу, которую открыл для работы какой-либо пользователь на том же компьютере или в сети. Удаление мгновенного снимка базы данных никак не влияет на базу данных-источник.

При удалении базы она удаляется из списка баз данных экземпляра сервера. Также с внешних носителей физически удаляются все файлы, относящиеся к этой базе данных — все файлы данных (первичный и вторичные) и все файлы журналов транзакций. Однако физическое удаление файлов происходит только в том случае, если база данных в момент удаления находится в состоянии `ONLINE`. Если же база неактивна (находится в состоянии `OFFLINE`), то удаление файлов не происходит.

При удалении мгновенного снимка сведения о нем удаляются из системного каталога и удаляются файлы мгновенного снимка.

* * *

Сейчас мы с вами создадим несколько простых баз данных с использованием оператора `CREATE DATABASE` при помощи утилиты `sqlcmd` и в программе `Management Studio`, но вначале потренируемся в отображении существующих баз данных, их файлов и интересующих нас характеристик — как характеристик баз данных, так и характеристик соответствующих файлов.

3.4.1.3. Создание и отображение баз данных в командной строке

Независимо от того, собираетесь ли вы когда-нибудь работать с командной строкой, рассмотрите детально следующие примеры. Они вам пригодятся и для работы в нормальной графической среде тоже.

Универсальной утилитой командной строки, позволяющей выполнять операторы `Transact-SQL` в `SQL Server`, является `sqlcmd`. Описание наиболее часто используемых параметров этой утилиты см. в *приложении 3*.

Для создания базы данных запустите на выполнение командную строку или `Windows PowerShell`. На экране появится соответствующее окно и подсказка, например:

```
PS C:\Users\Administrator>
```

В подсказке этого окна введите имя утилиты — `sqlcmd`. Если на вашем компьютере установлен единственный экземпляр сервера базы данных или вам требуется экземпляр сервера по умолчанию, то при вызове утилиты можно не задавать больше никаких параметров. На моем компьютере установлено два экземпляра SQL Server.

Для вызова утилиты, которая должна будет работать с версией Express Edition, требуется задать параметр `-S`, в котором нужно указать имя сервера и через обратную наклонную черту имя экземпляра сервера:

```
sqlcmd -S DEVRACE\SQLEXPRESS
```

Для вызова утилиты, работающей с экземпляром сервера по умолчанию, можно сделать так:

```
sqlcmd -S DEVRACE
```

Сам экземпляр сервера, компонент Database Engine, должен в момент вызова утилиты уже выполняться.

На экране появится подсказка самой утилиты:

```
1>
```

Давайте вначале отобразим существующие в экземпляре сервера базы данных, используя системное представление `sys.databases`. Введите в строке подсказки утилиты вначале команду `USE`, указывающую, что текущей базой данных является системная база данных `master`. После этой команды следует ввести `GO`, а затем оператор `SELECT`, обращающийся к системному представлению `sys.databases`. Нажмите клавишу `<Enter>` (пример 3.1).

Пример 3.1. Отображение баз данных текущего экземпляра сервера базы данных в системном представлении `sys.databases`

```
USE master;  
GO  
SELECT name, database_id, create_date, collation_name  
FROM sys.databases;
```

Однако ничего интересного не произойдет, только на экране появятся введенные строки и следующая строка подсказки утилиты с номером 4:

```
1> USE master;  
2> SELECT name, database_id, create_date, collation_name  
3> FROM sys.databases;  
4>
```

Для того чтобы был выполнен введенный оператор Transact-SQL (или группа операторов), нужно ввести еще и оператор `GO`. После ввода этого оператора и нажатия клавиши `<Enter>` появится список всех баз данных, существующих в экземпляре сервера. Если вы еще не создавали пользовательские базы данных, то список будет содержать только описания четырех системных баз данных: `master`, `tempdb`, `model` и `msdb` и двух баз данных, относящихся к компоненту Reporting Services и исполь-

зуемых для внутренних целей: ReportServer и ReportServerTempDB. Скрытая база данных resource не отображается в этом списке.

Оператор `SELECT`, используемый в этом примере, позволяет получить указанные данные из таблицы (таблиц) базы данных или из представления. Он также позволяет просто вывести заданные величины: любые литералы, константы, результаты обращения к различным функциям. В этом случае в операторе не задается предложение `FROM`.

В примере 3.1 после ключевого слова `SELECT` мы перечислили в списке выбора этого оператора имена тех характеристик (столбцов) представления, которые хотим отобразить, а в предложении `FROM` указали имя представления, из которого должны быть получены эти характеристики: `sys.databases`. В результате выполнения такого оператора мы получим список всех баз данных, существующих в текущем экземпляре сервера базы данных.

ЗАМЕЧАНИЕ

Следует напомнить, что ключевые слова языка Transact-SQL нечувствительны к регистру — их можно вводить как строчными, так и прописными буквами.

Однако полученный результат не производит хорошего впечатления. Сведения по каждой базе данных занимают несколько строк. Это сильно ухудшает читаемость результата. И дело не только в том, что по умолчанию длина строки в PowerShell составляет 120 символов, а в командной строке 80. Размер строки можно изменить, щелкнув правой кнопкой мыши по заголовку окна, выбрав элемент меню **Свойства** и изменив на вкладке **Расположение** размер окна по ширине. Неприятность в том, что размер отображаемых полей слишком велик.

Для улучшения читаемости результата внесем некоторые изменения в наш оператор `SELECT`, выполнив простые преобразования получаемых данных и задав осмысленные тексты заголовков отображаемых столбцов. В подсказке утилиты введите и выполните несколько измененный оператор выборки данных (пример 3.2):

Пример 3.2. Более правильное отображение в PowerShell баз данных текущего экземпляра сервера базы данных в системном представлении `sys.databases`

```
USE master;
GO
SELECT CAST(name AS CHAR(20)) AS 'NAME',
        CAST(database_id AS CHAR(4)) AS 'ID',
        create_date AS 'DATE',
        CAST(collation_name AS CHAR(23)) AS 'COLLATION'
FROM sys.databases;
GO
```

Теперь мы получили более симпатичный список баз данных. На моем ноутбуке этот список выглядит следующим образом:

NAME	ID	DATE	COLLATION
master	1	2003-04-08 09:13:36.390	Cyrillic_General_CI_AS
tempdb	2	2012-06-08 16:39:42.653	Cyrillic_General_CI_AS
model	3	2003-04-08 09:13:36.390	Cyrillic_General_CI_AS
msdb	4	2012-02-10 21:02:17.770	Cyrillic_General_CI_AS
ReportServer	5	2012-06-07 23:37:51.727	Latin1_General_CI_AS_KS
ReportServerTempDB	6	2012-06-07 23:37:53.120	Latin1_General_CI_AS_KS

(6 row(s) affected)

В операторе для трех столбцов мы используем весьма простую и очень удобную в работе функцию преобразования данных `CAST()` (которую довольно подробно с многочисленными примерами использования рассмотрим в следующей главе). Синтаксис функции прост:

```
CAST(<идентификатор> AS <тип данных>)
```

Эта функция позволяет привести тип данных заданной переменной или константы (параметр `идентификатор`) к типу данных, указанному после ключевого слова `AS` (параметр `тип данных`). Здесь мы используем эту функцию лишь с целью уменьшения размера поля, отводимого для отображения столбца. Например, имя базы данных может содержать до 128 символов. Если фактическое имя короче, то справа при его отображении система добавляет недостающие пробелы до максимального значения 128. Мы же в операторе при помощи функции `CAST()` сократили этот размер до 20 символов, преобразовав исходный строковый тип данных у столбца `name` (`CHAR(128)`) с количеством символов 128 опять же в строковый, но с другим размером, указав в выходном типе данных 20 символов (`CHAR(20)`).

Мы в операторе `SELECT` только не задали никакого преобразования для столбца, содержащего дату и время, поскольку преобразование, выполняемое по умолчанию при отображении этого типа данных, нас вполне устраивает.

После имени отображаемого столбца в операторе `SELECT` мы можем указать предложение `AS` (не путайте с параметром `AS` в функции преобразования данных `CAST()`) и задать в этом предложении в апострофах текст, который будет отображаться в заголовке соответствующего столбца. Именно это мы и сделали для каждого выбираемого столбца. Разумеется, здесь мы также можем задать и русскоязычные заголовки — 'Имя базы данных', 'Идентификатор', 'Дата и время создания', 'Порядок сортировки', хотя размер таких заголовков будет иметь несколько большую длину. Результирующий размер отображаемых столбцов будет соответствовать большему значению из заданного размера отображаемых данных и размера заголовка.

ЗАМЕЧАНИЕ ПО СИНТАКСИСУ

Если заголовок столбца в предложении `AS` не содержит специальных символов, в частности пробелов, как и в рассмотренном сейчас примере, а является правильным обычным идентификатором (см. главу 2), то его вообще-то можно было бы и не заключать в апострофы. В некоторых информационных и многих учебных материалах корпорации Microsoft, да и в документе Books Online, вы можете увидеть примеры использования такого варианта. Такая свобода задания строковых значений в данном случае в общем-то совершенно понятна. В этом месте ожидается использование

строкового данного, и соответствующий набор знаков воспринимается как строка символов. Наверное не стоит напоминать, что это не является хорошей практикой. Если мы используем строковые константы в наших пакетах, в данном случае при задании заголовков, то всегда следует заключать их в апострофы, чтобы избежать возможной путаницы, повысить читаемость кода и избежать лишних неприятностей при изменениях системы в будущем.

Несколько слов об операторе `GO`. Он не является оператором языка Transact-SQL, по этой причине он не может завершаться символом точка с запятой, наличие этого символа вызовет синтаксическую ошибку. Это служебный оператор утилиты `sqlcmd` и программы Management Studio. В SQL Server существует понятие пакета операторов (batch). Пакет содержит группу операторов Transact-SQL; фактическое выполнение группы начинается только после ввода оператора `GO`. Синтаксис оператора `GO`:

```
GO [<количество повторений>]
```

Если количество повторений (которое должно быть положительным целым числом) указано, то весь предыдущий фрагмент пакета выполняется заданное число раз. Если количество повторений не указано, то пакет выполняется один раз. Выполните предыдущий пакет операторов, указав оператор `GO` с любым количеством повторений, большим единицы. Вы получите заданное вами количество одинаковых наборов строк отображения баз данных, существующих в экземпляре сервера.

Теперь немного усложним оператор `SELECT`, указав, что в список вывода должны помещаться только сведения по базе данных `tempdb`. Рассмотрим один из простых вариантов предложения `WHERE` в операторе `SELECT`. Введите и выполните следующие операторы (пример 3.3).

Пример 3.3. Отображение одной базы данных в системном представлении sys.databases

```
USE master;
GO
SELECT CAST(name AS CHAR(20)) AS 'NAME',
       CAST(database_id AS CHAR(4)) AS 'ID',
       create_date AS 'DATE',
       CAST(collation_name AS CHAR(23)) AS 'COLLATION'
FROM sys.databases
WHERE name = 'tempdb';
GO
```

В предложении `WHERE` задается конкретное требуемое значение поля `name`, определяющее имя единственной отображаемой базы данных. В результат отображения попадет только одна заданная строка, описывающая базу данных `tempdb`:

NAME	ID	DATE	COLLATION
tempdb	2	2012-06-08 17:16:42.653	Cyrillic_General_CI_AS

(1 row(s) affected)

ЗАМЕЧАНИЕ

В самом начале каждого пакета мы записывали оператор `USE`, который указывает текущую базу данных, т. е. ту базу данных, с которой выполняются все последующие действия. Поскольку мы сейчас выполняем действия при обращении к системному представлению, которое присутствует в любой базе данных, как в системной, так и в пользовательской, то в данном случае не имеет значения, какая именно база данных является текущей. Оператор `USE` в этих примерах можно опустить. Во многих других случаях использования системных средств работы с базами данных имеет значение, какая база данных является текущей. Именно с базой данных, определенной в операторе `USE`, выполняются многие действия при вызове хранимых процедур, представлений или функций. Опять же по правилам хорошего тона следует всегда указывать этот оператор в ваших пакетах. Что лично я, к сожалению, делаю далеко не всегда. Очень рекомендую каждый раз после этого оператора вводить `GO`. В некоторых версиях системы отсутствие `GO` может приводить к неприятным результатам.

* * *

Здесь я хочу сделать небольшое отступление и сказать несколько слов об используемых в работе программных средствах и вообще об удобстве в работе. Утилиту `sqlcmd` можно запускать на выполнение в PowerShell и в обычной командной строке.

Результаты будут весьма похожими. Позже мы сможем сравнить вид этих отображений с тем, что можно получить в программе Management Studio. Надо сказать, что программисты имеют самые различные эстетические пристрастия. Кто-то предпочитает графический интерфейс (таких, разумеется, большинство), но есть люди, искренне любящие командную строку в различных ее вариантах и в принципе не признающие графический интерфейс. Правда, среди наших пользователей (как "юзеров", так и "ламеров") таких я что-то не встречал.

* * *

Более подробную информацию о базах данных и их файлах в текущем экземпляре сервера базы данных можно получить, используя системное представление просмотра каталогов `sys.master_files`. Введите и выполните следующий оператор, отображающий сведения о файлах баз данных (пример 3.4).

Пример 3.4. Отображение баз данных и их файлов в системном представлении `sys.master_files`

```
USE master;  
GO  
SELECT database_id, file_id, type, type_desc, data_space_id,  
       name, physical_name, is_read_only, state, state_desc,  
       size, max_size, growth, is_percent_growth  
FROM sys.master_files;  
GO
```

Вывод опять же будет не слишком наглядным. Здесь просто перечислены все те столбцы, которые мы с вами только что рассмотрели в предыдущем разделе.

Вначале нужно отобразить из представления те столбцы, которые могут быть интересны в первую очередь. Вот более хороший вариант отображения файлов баз данных в командной строке, пример 3.5.

Пример 3.5. Более правильный вариант отображения баз данных и их файлов в системном представлении sys.master_files

```
USE master;
GO
SELECT CAST(database_id AS CHAR(5)) AS 'DB ID',
       CAST(type_desc AS CHAR(6)) AS 'Descr',
       CAST(name AS CHAR(24)) AS 'File Name',
       CAST(state_desc AS CHAR(5)) AS 'State',
       CAST(size AS CHAR(5)) AS 'Size',
       max_size AS 'Max Size',
       CAST(growth AS CHAR(6)) AS 'Growth'
FROM sys.master_files;
GO
```

Результатом будет отображение всех файлов баз данных текущего экземпляра сервера с некоторыми их характеристиками:

DB ID	Descr	File Name	State	Size	Max Size	Growth
1	ROWS	master	ONLIN	624	-1	10
1	LOG	mastlog	ONLIN	224	-1	10
2	ROWS	tempdev	ONLIN	1024	-1	10
2	LOG	templog	ONLIN	64	-1	10
3	ROWS	modeldev	ONLIN	520	-1	128
3	LOG	modellog	ONLIN	128	-1	10
4	ROWS	MSDBData	ONLIN	2136	-1	10
4	LOG	MSDBLog	ONLIN	584	268435456	10
5	ROWS	ReportServer	ONLIN	648	-1	128
5	LOG	ReportServer_log	ONLIN	880	268435456	10
6	ROWS	ReportServerTempDB	ONLIN	520	-1	128
6	LOG	ReportServerTempDB_log	ONLIN	130	268435456	10

(12 row(s) affected)

По поводу красоты мы здесь вопрос вроде бы решили, однако возникают сомнения относительно безупречности полученного результата. Не очень нравятся какие-то числа в первом столбце этого результата: идентификаторы баз данных. Лучше было бы отображать здесь соответствующие имена баз данных.

Изменим обращение к системному представлению sys.master_files следующим образом, используя системную функцию DB_NAME(), позволяющую по идентификатору, получаемому из этого представления, находить имена баз данных. Простоты ради уберем некоторые столбцы. Выполните пример 3.6.

Пример 3.6. Отображение баз данных и их файлов в системном представлении sys.master_files

```
USE master;
GO
SELECT CAST(DB_NAME(database_id) AS CHAR(20)) AS 'DB Name',
        CAST(NAME AS CHAR(24)) AS 'File Name',
        CAST(state_desc AS CHAR(7)) AS 'State',
        CAST(type_desc AS CHAR(6)) AS 'Descr'
FROM sys.master_files
ORDER BY 'DB Name';
GO
```

Результатом будет следующий список:

DB Name	File Name	State	Descr
-----	-----	-----	-----
master	master	ONLINE	ROWS
master	mastlog	ONLINE	LOG
model	modeldev	ONLINE	ROWS
model	modellog	ONLINE	LOG
msdb	MSDBData	ONLINE	ROWS
msdb	MSDBLog	ONLINE	LOG
ReportServer	ReportServer	ONLINE	ROWS
ReportServer	ReportServer_log	ONLINE	LOG
ReportServerTempDB	ReportServerTempDB	ONLINE	ROWS
ReportServerTempDB	ReportServerTempDB_log	ONLINE	LOG
tempdb	tempdev	ONLINE	ROWS
tempdb	templog	ONLINE	LOG

(12 row(s) affected)

Благодаря использованию системной функции DB_NAME(), мы получили имена баз данных. Замечательно и то, что сделать это оказалось необыкновенно просто.

В этом пакете в операторе SELECT мы добавили еще одну возможность. Последней строкой оператора записано ORDER BY 'DB Name'. Это предложение позволяет упорядочить отображаемый список по значению первого поля, указанного в списке выбора оператора, что мы и видим в результате отображения. Причем мы указали не имя столбца, получаемого из системного представления, не его номер (эти варианты тоже возможны в операторе SELECT), а текст заголовка, заданный нами после ключевого слова AS в списке выбора.

Предложение ORDER BY можно задать и в виде ORDER BY 1. Здесь указывается, что упорядочение осуществляется по первому полю из списка выбора. Результат выполнения будет, разумеется, точно таким же.

Другое системное представление просмотра каталогов sys.database_files позволяет просмотреть все файлы одной текущей базы данных, заданной в операторе USE.

Введите и выполните следующие операторы для отображения файлов базы данных, скажем, master (пример 3.7).

Пример 3.7. Отображение файлов базы данных master в системном представлении sys.database_files

```
USE master;
GO
SELECT CAST(file_id AS CHAR(2)) AS 'ID',
        CAST(type AS CHAR(4)) AS 'Type',
        CAST(type_desc AS CHAR(11)) AS 'Description',
        CAST(name AS CHAR (12)) AS 'Name',
        state AS 'State',
        CAST(state_desc AS CHAR(10)) AS 'State desc',
        CAST(size AS CHAR(5)) AS 'Size'
FROM sys.database_files;
GO
```

Результат будет следующим:

ID	Type	Description	Name	State	State desc	Size
--	----	-----	-----	-----	-----	-----
1	0	ROWS	master	0	ONLINE	624
2	1	LOG	mastlog	0	ONLINE	224

(2 row(s) affected)

В точности такой же результат мы можем получить, используя и системное представление sys.master_files при задании в операторе SELECT предложения WHERE, в котором будет указан требуемый идентификатор нужной нам базы данных. Для системной базы данных master, как мы можем видеть из листинга примера 3.2, этот идентификатор равен единице. Выполните операторы примера 3.8:

Пример 3.8. Отображение файлов базы данных master в системном представлении sys.master_files

```
USE master;
GO
SELECT CAST(file_id AS CHAR(2)) AS 'ID',
        CAST(type AS CHAR(4)) AS 'Type',
        CAST(type_desc AS CHAR(11)) AS 'Description',
        CAST(name AS CHAR (12)) AS 'Name',
        state AS 'State',
        CAST(state_desc AS CHAR(10)) AS 'State desc',
        CAST(size AS CHAR(5)) AS 'Size'
FROM sys.master_files
WHERE database_id = 1;
GO
```

В предложении `WHERE` указывается, что должны отображаться только те строки файлов, для которых идентификатор базы данных (`database_id`) равен единице, т. е. будут отображаться строки файлов, относящиеся к базе данных `master`.

Этот пример выглядит как-то не очень красиво. Получается, что для того чтобы узнать значение идентификатора базы данных `master`, нам нужно выполнить отображение всех баз данных (см. пример 3.2), найти в полученном списке значение идентификатора базы данных `master` и подставить это значение в предложение `WHERE`.

На самом деле здесь можно и в одном операторе осуществить поиск идентификатора нужной базы данных, используя оператор `SELECT`, который обращается к системному представлению `sys.databases`. Для этого в предыдущем примере предложение `WHERE` нужно записать в следующем виде:

```
WHERE database_id =
    ( SELECT database_id
      FROM sys.databases
      WHERE name = 'master' );
```

Внутренний оператор `SELECT` в этом предложении возвращает значение идентификатора (столбец `database_id`) базы данных `master`, которая задается при помощи указания имени этой базы данных (столбец `name`). Обратите внимание, что по правилам синтаксиса SQL этот внутренний оператор `SELECT` обязательно должен быть заключен в круглые скобки.

Есть еще более простой способ выполнить нужное нам отображение файлов базы данных `master`, используя системную функцию `DB_ID()`, которая возвращает идентификатор базы данных по ее имени. Эту функцию мы будем еще не один раз использовать в наших скриптах. Синтаксис функции:

```
DB_ID([<имя базы данных>])
```

Если указанная в параметре база данных отсутствует в системе, то функция вернет значение `NULL`.

Введите и выполните операторы примера 3.9.

Пример 3.9. Лучший вариант отображения файлов базы данных `master` в системном представлении `sys.master_files`

```
USE master;
GO
SELECT CAST(file_id AS CHAR(2)) AS 'ID',
       CAST(type AS CHAR(4)) AS 'Type',
       CAST(type_desc AS CHAR(11)) AS 'Description',
       CAST(name AS CHAR(12)) AS 'Name',
       state AS 'State',
       CAST(state_desc AS CHAR(10)) AS 'State desc',
       CAST(size AS CHAR(5)) AS 'Size'
```



```
FROM sys.master_files
WHERE database_id = DB_ID('master');
GO
```

Если в функции `DB_ID()` не указать необязательный параметр имя базы данных, то она вернет идентификатор текущей базы данных, которая была задана в последнем операторе `USE`.

ЗАМЕЧАНИЕ

Если вам нужно отобразить только сведения по одному из файлов базы данных (это показано в примерах Books Online), то в предложении `WHERE` оператора `SELECT` можно задать имя столбца `name` и после знака равенства в апострофах имя интересующего вас файла, например `name = 'master'`. В этом случае вы получите сведения только по файлу данных базы данных `master`. Однако если в системе (в текущем экземпляре сервера базы данных) у различных баз данных существуют файлы с тем же именем, то вы получите список всех таких файлов. Так что наш с вами вариант отображения файлов конкретной базы данных из примера 3.9 много лучше всех других.

Давайте еще кратко рассмотрим очень простую системную функцию `FILE_NAME()`, которая всего лишь возвращает логическое имя файла базы данных (файла данных или журнала транзакций) по идентификатору этого файла для текущей базы данных. Синтаксис обращения к функции:

```
FILE_NAME(<идентификатор файла>)
```

Идентификатором может быть любое число. Если в текущей базе данных существует файл с таким идентификатором, то функция вернет его логическое имя. Иначе функция возвращает значение `NULL`. В качестве идентификатора вы можете указать и дробное число. В этом случае дробная часть просто отбрасывается (округление не выполняется). Можно задать нулевое значение (напомню — файлы в базе данных нумеруются, начиная с единицы) и даже отрицательное значение; результат, возвращенный функцией при таких значениях параметра, будет `NULL`, ошибка сгенерирована не будет.

Выполните следующие операторы (пример 3.10).

Пример 3.10. Отображение логических имен файлов базы данных `master` в системной функции `FILE_NAME()`

```
USE master;
GO
SELECT DB_ID() AS 'ID',
       CAST(FILE_NAME(1) AS CHAR(10)) AS 'Файл 1',
       CAST(FILE_NAME(2) AS CHAR(10)) AS 'Файл 2',
       CAST(FILE_NAME(3) AS CHAR(10)) AS 'Ничего';
GO
```

Результатом будет:

ID	Файл 1	Файл 2	Ничего
1	master	mastlog	NULL

(1 row(s) affected)

Здесь в операторе `USE` указывается база данных `master`, к которой будут обращаться по умолчанию все следующие операторы. В операторе `SELECT` для имен логических файлов мы также выполняем преобразование данных, чтобы результат поместился в одну строку. По ходу дела в этом операторе мы отображаем и идентификатор текущей базы данных. В функции `DB_ID()` мы не указали никакого имени базы данных, поэтому функция вернет идентификатор текущей базы данных, т. е. `master`.

В четвертом столбце задается несуществующий у базы данных номер файла — 3. Результатом, как мы видим, будет значение `NULL`.

Обратите внимание, что в данном примере в операторе `SELECT` не указывается предположение `FROM`, т. е. не говорится, откуда должны получаться результаты — из какой таблицы, из какого представления. Это означает, что на выходе такого запроса будет ровно одна строка, содержащая перечисленные в списке выбора оператора `SELECT` значения, полученные при обращении к функциям.

Все наши операторы мы вводили руками в диалоговом режиме в подсказке утилиты (ну, если уж быть честным, я-то копировал заранее подготовленные мною тексты из электронного варианта этой книги и помещал их в подсказку утилиты). Если вы допустите какую-либо ошибку, то вам придется заново повторять почти все введенные данные. Утилита `sqlcmd` имеет параметр `-i`, который позволяет указать имя файла (файл скрипта), из которого утилита будет читать операторы. Можно поместить пакет операторов в файл, корректировать многократно и выполнять при вызове утилиты `sqlcmd`. Пример использования файла скрипта:

```
sqlcmd -i "D:\Ex3-10.sql"
```

Здесь в параметре `-i` указывается полный путь к файлу и имя файла скрипта.

Еще про один параметр утилиты `sqlcmd`. Если мы хотим, чтобы результат выполнения утилиты выводился не на монитор, а помещался в какой-либо файл, то при вызове утилиты нужно задать параметр `-o`, в котором указывается путь к файлу и имя файла, куда утилита будет выводить все результаты и диагностические сообщения. Имя этого параметра является чувствительным к регистру: вы должны ввести именно строчную букву `o`, а не прописную.

Пример использования этого параметра:

```
sqlcmd -o "D:\Result.txt"
```

Все выходные данные, создаваемые при выполнении утилиты, будут выводиться в файл `Result.txt` в корневом каталоге на диске `D:`. Если файл отсутствует на диске, то он будет создан. Если же файл уже существует, то новые строки будут добавляться в конец файла, не изменяя существующих в файле данных.

Напомню, что описание наиболее полезных параметров утилиты `sqlcmd` содержится в *приложении 2*.

* * *

Теперь, наконец, создадим в утилите `sqlcmd` несколько новых пользовательских баз данных. Потом их поудаляем, чтобы затем опять создать, но уже с использованием `Management Studio`.

Собственно говоря, различные варианты создания баз данных хорошо описаны в документе `Books Online`. Сейчас мы с вами выполним похожие действия.

Создадим базу данных, где все, что можно, будем устанавливать по умолчанию. Выполните следующие операторы примера 3.11.

Пример 3.11. Создание и отображение базы данных со всеми значениями по умолчанию

```
USE master;
GO
CREATE DATABASE SimpleDB;
GO
SELECT CAST(file_id AS CHAR(2)) AS 'ID',
       CAST(type AS CHAR(4)) AS 'Type',
       CAST(type_desc AS CHAR(11)) AS 'Description',
       CAST(name AS CHAR (12)) AS 'Name',
       state AS 'State',
       CAST(state_desc AS CHAR(10)) AS 'State desc',
       CAST(size AS CHAR(5)) AS 'Size'
FROM sys.master_files
WHERE database_id = DB_ID('SimpleDB');
GO
```

В результате выполнения оператора `CREATE DATABASE` будет создана база данных `SimpleDB`. Все ее характеристики устанавливаются по умолчанию. При выполнении оператора `SELECT` в данном пакете мы получим следующий список файлов и их характеристик этой базы данных:

ID	Type	Description	Name	State	State desc	Size
1	0	ROWS	SimpleDB	0	ONLINE	520
2	1	LOG	SimpleDB_log	0	ONLINE	130

(2 row(s) affected)

Чаще всего в процессе проектирования баз данных вы будете создавать базу дан-ных и ее объекты, некоторое время с гордостью любоваться результатами вашей деятельности, а затем с грустью замечать, что вы что-то не учли, что-то сделали неверно. Тогда вы начнете вносить изменения в ваши скрипты и вновь запускать их на выполнение. Как правило, база данных пересоздается вами заново. Если вы за-

будете перед этим удалить уже созданную и не совсем правильную базу данных, то получите сообщение об ошибке. Сейчас я повторно ввел эти же самые операторы из примера 3.11 и получил следующее сообщение:

```
Msg 1801, Level 16, State 3, Line 1
```

```
Database 'SimpleDB' already exists. Choose a different database name.
```

(Сообщение 1801, уровень 16, состояние 3, строка 1

База данных 'SimpleDB' уже существует. Выберите другое имя базы данных.)

Чтобы избежать таких неприятностей, настоятельно рекомендую использовать функцию `DB_ID()`, которую мы с вами уже применяли для определения идентификатора базы данных по ее имени.

В пакетах SQL Server допустимо использование и операторов ветвления, в частности, оператора `IF`, который, как и в обычных языках программирования, позволяет сделать некоторые проверки и на основании результата таких проверок выполнить различные действия. Сейчас мы его используем для проверки существования нашей базы данных, которую собираемся заново создать. Внесите следующие изменения в ваш пакет создания базы данных SimpleDB (пример 3.12).

Пример 3.12. Создание базы данных с удалением существующей "старой" базы данных

```
USE master;  
GO  
IF DB_ID('SimpleDB') IS NOT NULL  
    DROP DATABASE SimpleDB;  
GO  
CREATE DATABASE SimpleDB;  
GO
```

В операторе `IF` мы проверяем при помощи функции `DB_ID()` существование базы данных SimpleDB. Если база данных существует, то функция вернет целое число — идентификатор этой базы данных, и тогда будет выполнен оператор удаления базы данных: `DROP DATABASE`.

Если же в системе нет такой базы данных, то функция `DB_ID()` вернет `NULL`. В этом случае оператор удаления не будет выполняться.

ЗАМЕЧАНИЕ

Возможность использования операторов ветвления, операторов циклов и некоторых других в пакетах SQL Server является необыкновенно удобным средством. Не все системы управления базами данных имеют такую возможность. Пользуясь случаем, хочу от имени всего прогрессивного человечества поблагодарить корпорацию Microsoft за такое средство.

Теперь создадим базу данных, с которой мы будем работать на протяжении всей этой книги. Это BestDatabase, а имена двух ее файлов будут Winner с соответствующими расширениями. При создании базы данных мы явно укажем логические

имена файла данных и файла журнала транзакций и для них зададим все необходимые значения параметров. Выполните операторы примера 3.13.

Пример 3.13. Создание базы данных BestDatabase

```
USE master;
GO
IF DB_ID('BestDatabase') IS NOT NULL
    DROP DATABASE BestDatabase;
GO
CREATE DATABASE BestDatabase
ON PRIMARY (NAME = BestDatabase_dat,
    FILENAME = 'D:\BestDatabase\Winner.mdf',
    SIZE = 5 MB,
    MAXSIZE = UNLIMITED,
    FILEGROWTH = 1 MB)
LOG ON (NAME = BestDatabase_log,
    FILENAME = 'D:\BestDatabase\Winner.ldf',
    SIZE = 2 MB,
    MAXSIZE = 30 MB,
    FILEGROWTH = 1 MB);
GO
```

Оба файла базы данных — файл данных и файл журнала транзакций — располагаются на диске D: в каталоге **BestDatabase**. Напомню, что на соответствующем диске каталог с этим именем уже должен существовать, иначе при выполнении оператора создания базы данных вы получите сообщение об ошибке. Сами же файлы с такими именами должны отсутствовать в указанном каталоге.

Для файла данных в создаваемой базе данных установлен начальный размер 5 Мбайт, приращение указано 1 Мбайт, максимальный размер не ограничивается, следовательно, файл может расти до исчерпания объема дискового пространства или до 16 Тбайт.

Начальный размер файла журнала транзакций задается 2 Мбайта, максимальный размер 30 Мбайт, а квант увеличения размера — 1 Мбайт.

МАЛЕНЬКОЕ ЗАМЕЧАНИЕ ПО СИНТАКСИСУ

При указании размеров в операторе `CREATE DATABASE` единицы измерения могут записываться сразу же после числа, а могут помещаться и через один или более пробелов. Во втором случае запись выглядит, как мне кажется, много лучше. Надеюсь, вы обратили внимание, что во всех примерах единицы измерения заданы явно — там, где можно было бы опустить указание мегабайтов, ключевое слово `MB` все равно присутствует.

Если у вас еще остаются смутные сомнения в необходимости явного задания величин, для которых можно было бы использовать значения по умолчанию, попробуйте разобраться в скриптах, написанных для других систем управления базами данных, где использованы принятые именно *там* значения по умолчанию.

Следующий пример в принципе повторяет предыдущий, однако здесь мне хочется рассмотреть некоторые дополнительные полезные средства, используемые в пакетах SQL Server, — локальные переменные и оператор `EXECUTE`. Выполните следующий пакет (пример 3.14).

Пример 3.14. Создание базы данных BestDatabase, другой вариант

```
USE master;
GO
IF DB_ID('BestDatabase') IS NOT NULL
    DROP DATABASE BestDatabase;
GO

DECLARE @path AS VARCHAR(255),
        @path_data AS VARCHAR(255),
        @path_log AS VARCHAR(255);
SET @path = 'D:\BestDatabase\';
SET @path_data = @path + 'Winner.mdf';
SET @path_log = @path + 'Winner.ldf';

EXECUTE (
    'CREATE DATABASE BestDatabase
    ON PRIMARY (NAME = BestDatabase_dat,
        FILENAME = ''' + @path_data + ''',
        SIZE = 5 MB,
        MAXSIZE = UNLIMITED,
        FILEGROWTH = 1 MB)
    LOG ON (NAME = BestDatabase_log,
        FILENAME = ''' + @path_log + ''',
        SIZE = 2 MB,
        MAXSIZE = 30 MB,
        FILEGROWTH = 1 MB);');
GO
```

В этом примере мы в операторе `DECLARE` объявляем три локальные переменные, т. е. переменные, используемые только в данном пакете: `@path`, `@path_data` и `@path_log`, указав для них строковый тип данных переменной длины до 255 символов (`AS VARCHAR(255)`). Имена локальных переменных должны начинаться с символа `@`. Затем операторами `SET` мы присваиваем этим переменным значения, причем для переменных `@path_data` и `@path_log` мы используем операцию конкатенации (соединения строк), которая задается символом плюс (+). В результате эти две локальные переменные будут иметь значение полного пути к файлу данных и к файлу журнала транзакций соответственно.

ЗАМЕЧАНИЕ

Существование объявленных локальных переменных ограничено оператором `GO`. После выполнения этого оператора система уже ничего "не знает" про любые объявлен-

ные локальные переменные. Далее в скрипте можно объявлять переменные с теми же именами и с любыми иными типами данных.

Собственно для создания базы данных мы выполняем оператор `EXECUTE` (для него можно также использовать и сокращение `EXEC`), которому в качестве параметра передаем строку, которую создаем опять же при выполнении конкатенации строковых констант и значений локальных параметров `@path_data` и `@path_log`. Вся строка заключается в апострофы, а параметр оператора помещен в круглые скобки.

Обратите внимание, как здесь определяются имена файлов данных и журнала транзакций:

```
FILENAME = ''' + @path_data + ''',  
...  
FILENAME = ''' + @path_log + ''',  
...
```

После знака равенства подряд идут три апострофа. Первые два задают апостроф *внутри* предыдущей части строки (вы помните, что для представления одного апострофа в строке, заключенной в апострофы, нужно записать подряд два апострофа). Третий апостроф завершает предыдущую строку.

Похожим образом в следующей группе из трех апострофов первый начинает строку, а другие два задают апостроф *внутри* этой строки. В результате выполнения всех этих действий мы получаем пути к файлам, которые по правилам синтаксиса должны быть заключены в апострофы:

```
'D:\BestDatabase\Winner.mdf'
```

и

```
'D:\BestDatabase\Winner.ldf'
```

Вся созданная таким образом строка является правильным оператором `CREATE DATABASE`, который задает создание новой базы данных.

Для того чтобы просмотреть и проверить правильность результата формирования такой строки, достаточно в этом пакете только лишь заменить оператор `EXECUTE` на `SELECT`. Строка будет отображена на мониторе. Здесь довольно легко можно найти и исправить ошибки. Что я и сделал при написании предыдущего примера, поскольку вначале при создании этой строки допустил ошибку.

Пример 3.14 является не просто демонстрацией некоторых из тех возможностей, которые существуют в SQL Server. Этот прием позволяет в программных пакетах на основании каких-то условий динамически создавать операторы Transact-SQL. Есть еще один способ динамического создания операторов с использованием утилиты `sqlcmd`, который мы рассмотрим чуть позже.

Теперь создадим базу данных, содержащую два файла данных и два файла журнала транзакций. По правде сказать, практически ничего нового мы здесь не увидим (пример 3.15). Напомню только, что перед выполнением примера на диске `D:` нужно создать каталог `Multy`.

Пример 3.15. Создание многофайловой базы данных

```
USE master;
GO
IF DB_ID('Multy') IS NOT NULL
    DROP DATABASE Multy;
GO
CREATE DATABASE Multy
ON
PRIMARY
    ( NAME = Multy1,
      FILENAME = 'D:\Multy\Multy1.mdf' ),
    ( NAME = Multy2,
      FILENAME = 'D:\Multy\Multy2.ndf' )
LOG ON
    ( NAME = MultyL1,
      FILENAME = 'D:\Multy\MultyL1.ldf' ),
    ( NAME = MultyL2,
      FILENAME = 'D:\Multy\MultyL2.ldf' );
GO
```

Думаю, здесь нам с вами все понятно. Все заданные характеристики в операторе создания базы данных нам уже известны. Следует только напомнить, что создание нескольких файлов журнала транзакций на одном и том же носителе не является осмысленным занятием. Более одного журнала транзакций следует создавать на различных носителях, если есть проблемы с доступным объемом внешней памяти.

* * *

Во всех предыдущих примерах создаваемые базы данных имели только одну файловую группу — первичную (PRIMARY), которая обязательно присутствует для каждой базы данных. База данных помимо первичной файловой группы может содержать и произвольное количество других файловых групп, называемых пользовательскими или вторичными файловыми группами.

Теперь создадим базу данных, в которой будет две файловые группы, каждая из которых содержит, скажем, по два файла данных.

Рассмотрим фрагмент синтаксиса оператора CREATE DATABASE, предложение FILEGROUP, относящееся к созданию файловых групп:

```
<файловая группа> ::=
    FILEGROUP <имя файловой группы> [ CONTAINS FILESTREAM ] [DEFAULT]
    <спецификация файла> [, <спецификация файла>]...
```

Для файловой группы указывается имя, которое должно быть уникальным в текущей базе данных, после чего следует описание как минимум одного файла данных.

Для создания такой базы данных с двумя файловыми группами выполните операторы примера 3.16.

Пример 3.16. Создание базы данных с двумя файловыми группами

```
USE master;
GO
IF DB_ID('MultyGroup') IS NOT NULL
    DROP DATABASE MultyGroup;
GO
CREATE DATABASE MultyGroup
ON
PRIMARY
    ( NAME = MultyGroup1,
      FILENAME = 'D:\MultyGroup\MultyGroup1.mdf'),
    ( NAME = MultyGroup2,
      FILENAME = 'D:\MultyGroup\MultyGroup2.ndf'),
FILEGROUP MultyGroup2
    ( NAME = MultyGroup3,
      FILENAME = 'D:\MultyGroup\MultyGroup3.ndf'),
    ( NAME = MultyGroup4,
      FILENAME = 'D:\MultyGroup\MultyGroup4.ndf')
LOG ON
    ( NAME = MultyGroupLog1,
      FILENAME = 'D:\MultyGroup\MultyGroupLog1.ldf'),
    ( NAME = MultyGroupLog2,
      FILENAME = 'D:\MultyGroup\MultyGroupLog2.ldf');
GO
```

Вообще-то все основные характеристики файлов данных и журнала транзакций взяты с некоторыми изменениями из примера 3.15. Здесь только добавлена вторичная файловая группа с именем `MultyGroup2`.

Теперь посмотрим, что у нас в результате получилось. Сначала отобразим файлы созданной базы данных. Обратимся к уже хорошо знакомому нам системному представлению `sys.database_files`. Выполните оператор примера 3.17.

Пример 3.17. Отображение состояния базы данных с двумя файловыми группами

```
USE MultyGroup;
GO
SELECT CAST(file_id AS CHAR(2)) AS 'ID',
       CAST(type AS CHAR(4)) AS 'Type',
       CAST(type_desc AS CHAR(11)) AS 'Description',
       CAST(name AS CHAR (16)) AS 'Name',
       state AS 'State',
       CAST(state_desc AS CHAR(10)) AS 'State desc'
FROM sys.database_files;
GO
```

Будет получен следующий результат:

ID	Type	Description	Name	State	State desc
1	0	ROWS	MultyGroup1	0	ONLINE
2	1	LOG	MultyGroupLog1	0	ONLINE
3	0	ROWS	MultyGroup2	0	ONLINE
4	0	ROWS	MultyGroup3	0	ONLINE
5	0	ROWS	MultyGroup4	0	ONLINE
6	1	LOG	MultyGroupLog2	0	ONLINE

(6 row(s) affected)

Похожий результат можно получить, как вы помните, и при использовании системного представления `sys.master_files`.

Теперь отобразите сведения только по файловым группам, используя системное представление `sys.filegroups`, как это показано в примере 3.18.

Пример 3.18. Отображение файловых групп базы данных MultyGroup

```
USE MultyGroup;
GO
SELECT CAST(name AS CHAR(12)) AS 'Name',
       CAST(type AS CHAR(2)) AS 'Type',
       CAST(type_desc AS CHAR(16)) AS 'Descripriion',
       CAST(is_read_only AS CHAR(1)) AS 'Read-only'
FROM sys.filegroups;
GO
```

Результат:

Name	Type	Descripriion	Read-only
PRIMARY	FG	ROWS_FILEGROUP	0
MultyGroup2	FG	ROWS_FILEGROUP	0

(2 row(s) affected)

Результат, надо сказать, малоинформативный. Здесь мы можем увидеть только имена файловых групп. Впрочем, чаще всего и этого бывает вполне достаточно.

В завершение использования утилиты командной строки `sqlcmd` мне хочется показать вам одну возможность применения параметров при выполнении в этой утилите заранее подготовленного скрипта.

Ранее мы сказали несколько слов о том, что при вызове утилиты можно не вводить в диалоговом режиме все нужные операторы, а поместить пакет соответствующих операторов в файл скрипта и при вызове утилиты указать имя этого скрипта при использовании параметра утилиты `-i`.

Скрипт может содержать параметры, значения которым присваиваются при вызове утилиты. Структура имени параметра следующая:

\$ (<имя параметра>)

Задание значений параметрам в скрипте выполняется при использовании параметра (или, иными словами, переключателя) утилиты `-v`. Здесь после имени переключателя указывается имя параметра, знак равенства и в кавычках значение параметра.

Рассмотрим пример скрипта с параметрами. Подготовим скрипт, содержащий оператор создания простой базы данных. Имена базы данных, файлов и каталогов для хранения файлов базы данных зададим при помощи параметра \$ (DBNM).

Создайте, например, в Блокноте следующий скрипт:

Пример 3.19. Скрипт создания базы данных, содержащий параметр

```
USE master;
GO
IF DB_ID('$ (DBNM) ') IS NOT NULL
    DROP DATABASE $ (DBNM) ;
GO
CREATE DATABASE $ (DBNM)
ON PRIMARY (NAME = $ (DBNM) _dat,
    FILENAME = 'D:\$ (DBNM) \$ (DBNM) .mdf',
    SIZE = 5 MB,
    MAXSIZE = UNLIMITED,
    FILEGROWTH = 1 MB)
LOG ON (NAME = $ (DBNM) _log,
    FILENAME = 'D:\$ (DBNM) \$ (DBNM) .ldf',
    SIZE = 2 MB,
    MAXSIZE = 30 MB,
    FILEGROWTH = 1 MB);
GO
SELECT CAST(file_id AS CHAR(2)) AS 'ID',
    CAST(type AS CHAR(4)) AS 'Type',
    CAST(type_desc AS CHAR(11)) AS 'Description',
    CAST(name AS CHAR (12)) AS 'Name',
    CAST(size AS CHAR(5)) AS 'Size'
FROM sys.master_files
WHERE database_id = DB_ID('$ (DBNM) ');
GO
```

Сохраните этот скрипт в корневом каталоге на диске D: с именем, например, `CreateDBParam.sql`.

Здесь параметр \$ (DBNM) используется для задания имени базы данных, имени каталога на диске D: для размещения файлов создаваемой базы данных, для задания логических имен файла данных и файла журнала транзакций и для задания имен

файлов в операционной системе. При задании путей к файлам, логических и физических имен файлов используется конкатенация, соединение нескольких строк. Причем никаких знаков операции для конкатенации в этом случае не требуется. Параметр \$(DBNM) просто будет заменен заданным при вызове утилиты значением.

Создайте на диске D: каталог с именем DBParam. Выполните утилиту в командной строке (предварительно завершив выполнение предыдущего сеанса утилиты, введя оператор quit):

```
sqlcmd -i "D:\CreatedBParam.sql" -v DBNM="DBParam"
```

В результате будет создана новая база данных. На мониторе отображается результат выполнения скрипта:

ID	Type	Description	Name	Size
1	0	ROWS	DBParam_dat	640
2	1	LOG	DBParam_log	256

(2 row(s) affected)

Посмотрите, как выполняется подстановка указанного при запуске утилиты значения параметра. Вот условный оператор в скрипте, осуществляющий проверку существования базы данных и при необходимости ее удаление:

```
IF DB_ID('$(DBNM)') IS NOT NULL
    DROP DATABASE $(DBNM);
```

В процессе выполнения скрипта этот оператор после подстановки значения параметра будет выглядеть следующим образом:

```
IF DB_ID('DBParam') IS NOT NULL
    DROP DATABASE DBParam;
```

Аналогичным образом будут выполнены подстановки значения параметра и в предложениях других операторов. Вот как будет выполнена конкатенация значения параметра со строками в предложении FILENAME, задающем имя файла данных. Исходное предложение:

```
FILENAME = 'D:\$(DBNM)\$(DBNM).mdf',
```

После подстановки значения эта строка примет вид:

```
FILENAME = 'D:\DBParam\DBParam.mdf',
```

Скрипт может содержать произвольное количество параметров. Синтаксис задания им значений при вызове утилиты является довольно свободным. Значения параметрам задаются после переключателя (иногда переключатель называют параметром — здесь не смешивайте терминологию) утилиты -v. Присваивание значений параметрам может следовать после этого переключателя, отделяясь друг от друга пробелами, либо перед каждым присваиванием можно указывать переключатель утилиты -v.

Например, если в скрипте присутствует три параметра: $\$(P1)$, $\$(P2)$ и $\$(P3)$, то значения им можно задать в виде

```
sqlcmd ... -v P1="V1" P2="V2" P3="V3"
```

или в следующей форме:

```
sqlcmd ... -v P1="V1" -v P2="V2" -v P3="V3"
```

В любом случае при вызове утилиты значения должны быть заданы *всем* параметрам, присутствующим в выполняемом скрипте.

* * *

Надеюсь, вы получили соответствующее удовольствие от работы с утилитой командной строки. В дальнейшем я буду описывать работу с операторами языка Transact-SQL, не привязываясь к средствам реализации. Где вы их будете применять — это ваше решение. Совершенно одинаково (или почти одинаково) эти операторы можно выполнять как при вызове утилиты `sqlcmd` в командной строке или в PowerShell, так и в компоненте системы с более мощными и удобными в работе возможностями: Management Studio. В Management Studio вам, скорее всего, не потребуется выполнение преобразований `CAST()`, как мы это делали в предыдущих примерах, отображая сведения о базах данных и их файлах, поскольку этот компонент автоматически определяет размер каждого столбца. Если такие размеры вас не устраивают, то вы легко с помощью мыши можете их изменять.

Сейчас мы перейдем к рассмотрению средств, существующих в Management Studio.

3.4.1.4. Создание и отображение баз данных в Management Studio

Программа Management Studio является очень мощным и удобным средством работы с базами данных в SQL Server.

Запустите на выполнение Management Studio. Щелкните мышью по кнопке **Пуск**, выберите **Все программы**, щелкните по строке **Microsoft SQL Server 2012** и затем по элементу **SQL Server Management Studio**. Появится диалоговое окно соединения с сервером. В окне **Connect to Server** можно выбрать тип сервера **Server type**, имя сервера **Server name** (имя одного из экземпляров сервера, установленных на компьютере — если существует несколько серверов), вид аутентификации **Authentication** (выберите из раскрывающегося списка **Windows Authentication**) и ввести имя пользователя **User name** (рис. 3.6).

Если на компьютере установлен только один экземпляр сервера базы данных, то в этом окне нужно, ничего не изменяя и не выбирая, просто щелкнуть по кнопке **Connect** (соединиться). Произойдет соединение с текущим экземпляром Database Engine и следом появится главное окно Management Studio, показанное на рис. 3.7.

Однако если на вашем компьютере установлено несколько серверов базы данных, то вам нужно из раскрывающегося списка **Server name** выбрать соответствующий сервер. У меня установлено два сервера, и каждый раз при запуске Management Studio я стараюсь вспомнить, для какого сервера и для каких целей я это делаю.



Рис. 3.6. Соединение с сервером

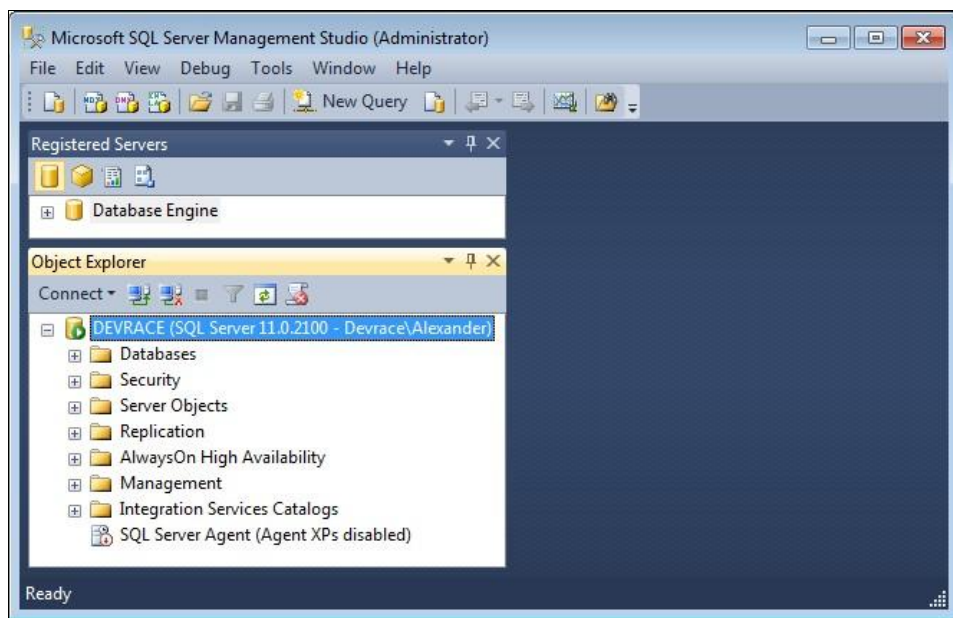


Рис. 3.7. Главное окно Management Studio. Панель **Object Explorer**

Если в окне не виден **Object Explorer**, выберите в меню **View | Object Explorer** или нажмите клавишу <F8>.

Как обычно, в верхней части окна располагается главное меню, ниже присутствует панель инструментов с кнопками быстрого доступа к функциям некоторых элементов главного меню. В левой части находится Инспектор объектов (**Object Explorer**), при помощи которого вы можете получить быстрый доступ ко многим полезным возможностям Management Studio, большинство из которых мы вскоре рассмотрим.

Чтобы выполнять в Management Studio пакеты операторов Transact-SQL, необходимо вызвать окно выполнения запросов. Для этого на панели инструментов нужно мышью щелкнуть по кнопке **New Query** (новый запрос).

Для вызова этого окна можно также щелкнуть мышью по элементу меню **File** (файл), выбрать элемент **New** (новый), а затем элемент **Query with Current Connection** (запрос в текущем соединении). В дальнейшем подобный вызов элемента в многоуровневом меню мы будем изображать в тексте чуть короче в следующем виде: "Выберите меню **File | New | Query with Current Connection**".

Наконец, для вызова окна запросов можно просто нажать клавиши <Ctrl>+<N>.

В центральной части главного окна появится пустое окно запросов, где можно вводить, изменять и выполнять операторы Transact-SQL. Это окно показано на рис. 3.8.

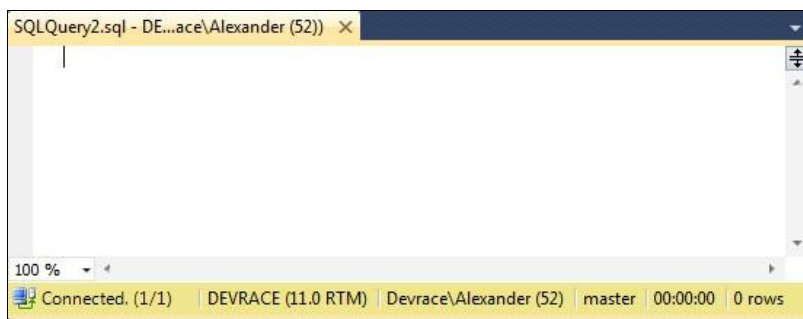


Рис. 3.8. Окно ввода операторов Transact-SQL

Вы можете создать произвольное количество окон запросов, в которых могут содержаться любые операторы работы с базами данных. Окна запросов создаются в виде вкладок. Переключение между этими вкладками выполняется щелчком мыши по заголовкам вкладок.

Тренировки ради предлагаю повторить некоторые скрипты, которые вы вводили в утилите командной строки чуть раньше в этой главе. Например, повторите создание базы данных SimpleDB, введя следующие операторы (пример 3.20).

Пример 3.20. Создание и отображение базы данных со значениями по умолчанию

```
USE master;
GO
IF DB_ID('SimpleDB') IS NOT NULL
    DROP DATABASE SimpleDB;
CREATE DATABASE SimpleDB;
GO
SELECT file_id AS 'ID',
       type AS 'Type',
       type_desc AS 'Description',
       name AS 'Name',
```

```

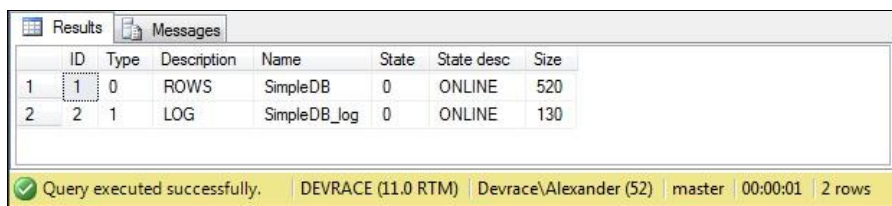
state AS 'State',
state_desc AS 'State desc',
size AS 'Size'
FROM sys.master_files
WHERE database_id = DB_ID('SimpleDB');
GO

```

Программа Management Studio при вводе операторов SQL выполняет выделение цветом ключевых слов, констант, подчеркивает красной волнистой линией неверные выражения. Иными словами, помогает вам создать правильный текст. Вы сможете выявить ошибки еще до того, как запустите скрипт на выполнение. Правда, бывают и такие случаи, когда красной волнистой линией подчеркиваются как бы ошибочные тексты, которые на самом деле созданы правильно.

Для выполнения этих введенных операторов щелкните мышью по кнопке **Execute** (выполнить) на панели инструментов, выберите в меню **Query** (запрос) | **Execute** (выполнить) либо нажмите клавишу <F5> или комбинацию клавиш <Ctrl>+<E>.

Результат выполнения пакета примера 3.20 показан на рис. 3.9. Это окно появится в нижней части окна выполнения запросов.



	ID	Type	Description	Name	State	State desc	Size
1	1	0	ROWS	SimpleDB	0	ONLINE	520
2	2	1	LOG	SimpleDB_log	0	ONLINE	130

Query executed successfully. | DEVRACE (11.0 RTM) | Devrace\Alexander (52) | master | 00:00:01 | 2 rows

Рис. 3.9. Результат создания простой базы данных

Заметьте, что в отличие от примера 3.9, где эта база данных создавалась и отображалась в командной строке, нам в данном случае при отображении результата, показанного на рис. 3.9, не нужно рассчитывать размер и количество символов, которое уместится на выходе. Нет необходимости выполнять преобразования отображаемых столбцов с использованием функции `CAST()`. В полученном окне результата мы всегда легко с помощью мыши можем уменьшить или расширить поле, отводимое для отображения любого столбца, если система не даст нам приличного варианта. Для этого нужно курсор мыши подвести к границе двух столбцов в заголовке и, нажав левую кнопку, изменить требуемый размер. Как правило, программа Management Studio с самого начала предоставляет хорошо читаемый вариант.

Вы можете заранее любыми средствами (пусть даже в программе Блокнот) создать файл скрипта, который будет содержать все необходимые операторы. Принято таким файлам давать расширение `sql`, хотя это также не является обязательным требованием к скриптам. Чтобы загрузить такой файл в окно запросов Management Studio, нужно выбрать в меню **File** | **Open** | **File**. Можно нажать клавиши <Ctrl>+<O> или щелкнуть мышью по кнопке открытия на панели инструментов.

Появится обычное окно открытия файла, в котором вы выбираете нужный для работы скрипт. Программа создаст новую вкладку и поместит туда выбранный текст.

Если вы создавали скрипт в Management Studio или вносили изменения в существующий скрипт, то вы можете сохранить эти изменения, выбрав в меню **File | Save (имя скрипта)**, нажав комбинацию клавиш <Ctrl>+<S> или щелкнув мышью по кнопке сохранения на инструментальной панели.

Если вы создавали новый скрипт или хотите сохранить существующий скрипт на диске с другим именем, то для этого следует выбрать в меню **File | Save [существующее имя скрипта] As ...** и в появившемся диалоговом окне сохранения файла выбрать каталог размещения и новое имя скрипта.

Программа Management Studio предоставляет еще одну удобную возможность. Если в окне существует множество операторов, а вам нужно выполнить только некоторые из них, то достаточно при помощи мыши или клавиатуры выделить требуемую последовательную группу операторов и запустить на выполнение только их. Если вы когда-либо присутствовали на мероприятиях, где специалисты Microsoft рассказывали что-нибудь интересное о возможностях системы с демонстрацией этих возможностей при использовании Management Studio, то вы, конечно же, видели, что они постоянно используют этот прием.

Теперь в Management Studio создадим базу данных BestDatabase, которую мы будем использовать в дальнейшей работе. Введите следующие операторы (пример 3.21).

Пример 3.21. Создание базы данных BestDatabase в Management Studio

```
USE master;
GO
IF DB_ID('BestDatabase') IS NOT NULL
    DROP DATABASE BestDatabase;
GO
CREATE DATABASE BestDatabase
ON PRIMARY (NAME = BestDatabase_dat,
    FILENAME = 'D:\BestDatabase\Winner.mdf',
    SIZE = 5 MB,
    MAXSIZE = UNLIMITED,
    FILEGROWTH = 1 MB)
LOG ON (NAME = BestDatabase_log,
    FILENAME = 'D:\BestDatabase\Winner.ldf',
    SIZE = 2 MB,
    MAXSIZE = 30 MB,
    FILEGROWTH = 1 MB);
GO
```

Выполните операторы.

А для отображения созданной базы данных в этой среде в отличие от утилиты `sqlcmd` можно поступить несколько иначе, нет необходимости использовать только оператор `SELECT` и системное представление `sys.master_files`. Щелкните мышью по символу (+) слева от строки **Databases** в окне **Object Explorer**. Раскроется список баз данных, определенных в текущем экземпляре сервера базы данных (рис. 3.10). Имена баз данных в списке упорядочены по алфавиту. Только хочу напомнить, что те средства, которые вы использовали в утилите `sqlcmd` для отображения характеристик баз данных и их файлов, можно с тем же успехом использовать в Management Studio.

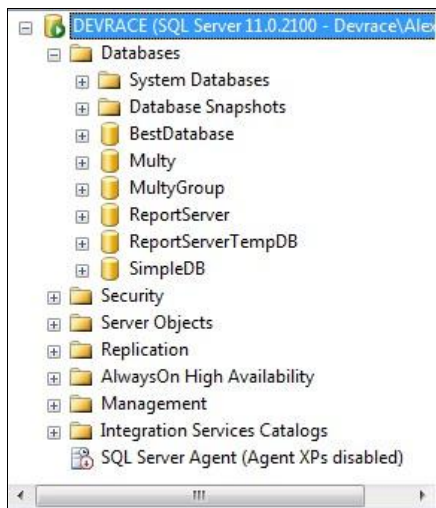


Рис. 3.10. Список баз данных в Object Explorer

Мы видим в этом списке и созданную только что базу данных **BestDatabase**.

ЗАМЕЧАНИЕ

Если вновь созданная база данных в окне **Object Explorer** не видна (а скорее всего так и будет сразу после создания новой базы данных), то следует обновить список объектов, щелкнув правой кнопкой мыши по строке **Databases** или по имени сервера базы данных и выбрав в появившемся контекстном меню элемент **Refresh** (обновить). Для обновления списка также можно, как и в любом другом приложении Windows, просто нажать клавишу <F5>. Не забудьте только в этом случае фокус перевести именно на **Object Explorer**, щелкнув мышью по заголовку этого окна, и выделить мышью строку **Databases** или строку сервера (самую первую строку в списке). Иначе у вас просто запустится на выполнение скрипт из текущего окна запросов.

Чтобы просмотреть подробнейшие сведения о только что созданной базе данных **BestDatabase** и при желании внести некоторые изменения, щелкните правой кнопкой мыши по строке **BestDatabase** и в появившемся контекстном меню выберите элемент **Properties** (свойства). Откроется окно свойств выбранной базы данных, где текущей будет вкладка **General** (общие) (рис. 3.11).

Здесь мы видим общие свойства базы данных: имя базы, дату создания, порядок сортировки, владельца и ряд других свойств. На этой вкладке не допускается внесение каких-либо изменений.

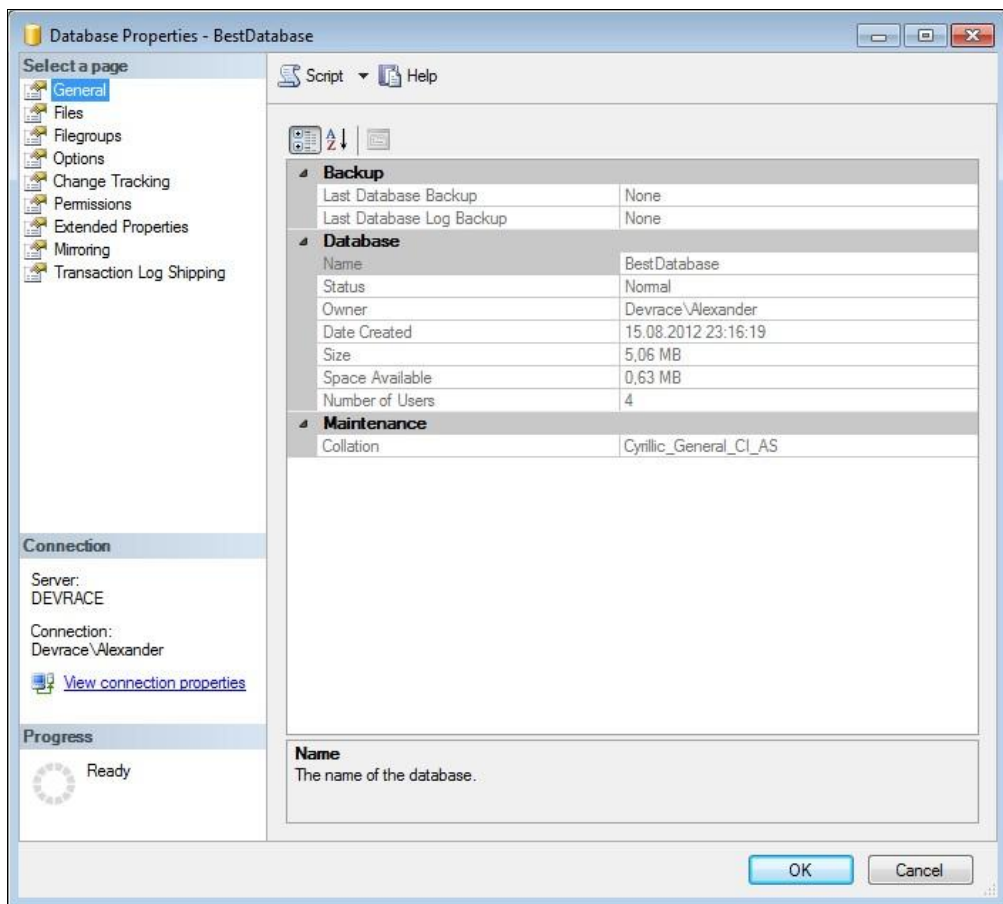


Рис. 3.11. Свойства базы данных BestDatabase. Вкладка **General**

В левой верхней части главного окна, в панели **Select a page** (выбор страницы) щелкните мышью по строке **Files** (файлы).

Откроется более интересная вкладка **Files**, где присутствует описание характеристик всех файлов, входящих в состав этой базы данных. Вкладка показана на рис. 3.12.

Здесь мы видим характеристики двух созданных файлов базы данных — файла данных с логическим именем `BestDatabase_dat` и файла журнала транзакций с логическим именем `BestDatabase_log`. Указаны их типы: `Rows Data` (файл данных, дословно — строки данных) и `Log` (журнал транзакций). Для них представлены начальный размер в мегабайтах, порядок увеличения размера, путь к файлу и имя файла (на этом рисунке не видно, однако можно просмотреть, воспользовавшись в нижней части окна полосой прокрутки). Более подробно отображаемые характеристики файлов при полностью развернутом окне свойств базы данных показаны на рис. 3.13.

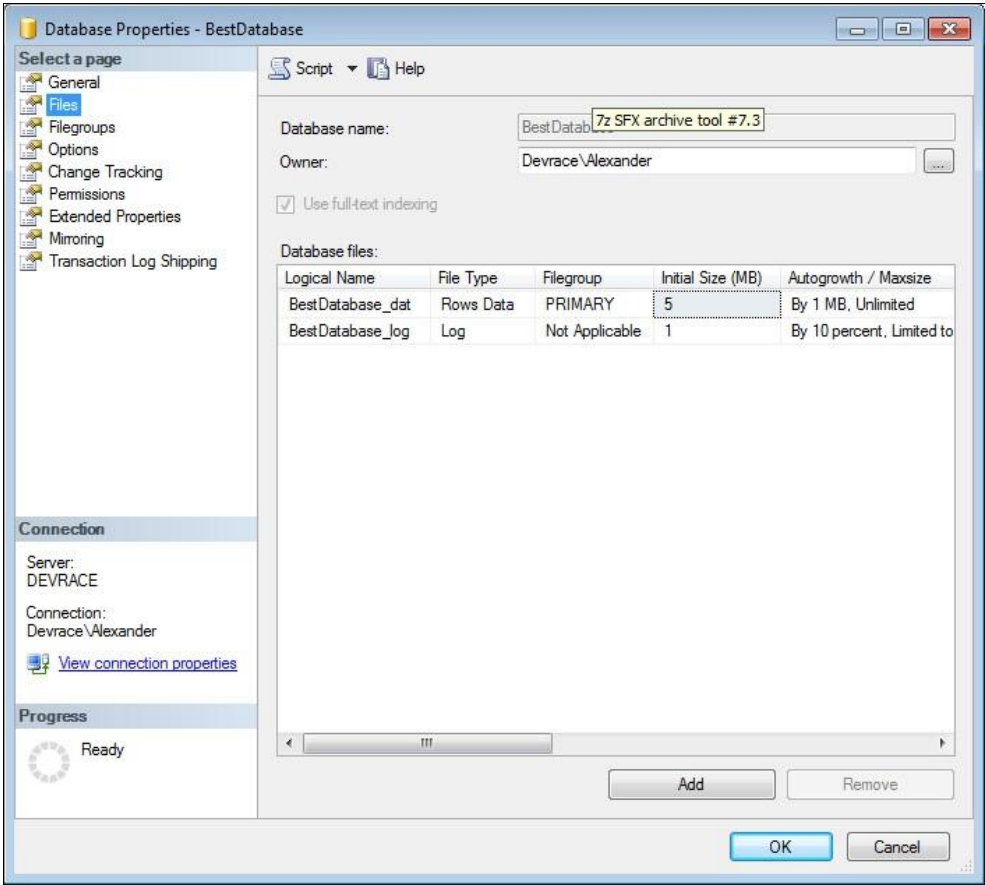


Рис. 3.12. Свойства базы данных BestDatabase. Вкладка Files

Logical Name	File Type	Filegroup	Initial Size (MB)	Autogrowth / Maxsize	Path
BestDatabase_dat	Rows Data	PRIMARY	5	By 1 MB, Unlimited	D:\BestDatabase
BestDatabase_log	Log	Not Applicable	1	By 10 percent, Limited to 209...	D:\BestDatabase

Рис. 3.13. Характеристики файлов базы данных BestDatabase

Давайте подробнее рассмотрим этот список.

В столбце **Logical Name** (логическое имя) мы видим логические имена двух созданных файлов — файла данных (BestDatabase_dat) и файла журнала транзакций (BestDatabase_log).

В столбце **File Type** (тип файла) указывается именно тип файла — файл данных (Rows Data) или файл журнала транзакций (Log).

Столбец **Filegroup** содержит имя файловой группы, которой принадлежит соответствующий файл. Для файлов данных в этом столбце указывается имя файловой группы. Здесь для первичного файла данных указано PRIMARY, т. е. файл относится

к первичной файловой группе. Для файлов же журнала транзакций здесь содержится текст *Not Applicable* (не применимо), поскольку для файлов журнала транзакций не применяется распределение по файловым группам.

Столбец **Initial Size (MB)** указывает начальный размер файла в мегабайтах.

В столбце **Autogrowth** (автоматическое увеличение размера) описывается единица увеличения размера памяти (единица измерения или проценты от начального значения) и до какой величины может увеличиваться размер памяти, отводимой под файл, либо указывается, что размер не ограничивается (*unrestricted growth*).

Столбец **Path** (путь) содержит полный путь к файлу.

В столбце **File Name** содержится имя физического файла в операционной системе.

При выборе на панели **Select a page** вкладки **Options** (свойства) появится окно свойств базы данных (рис. 3.14).

Здесь перечисляется множество общих свойств базы данных. Эти свойства, характеристики и возможность изменения отдельных значений мы рассмотрим чуть позже в этой главе.

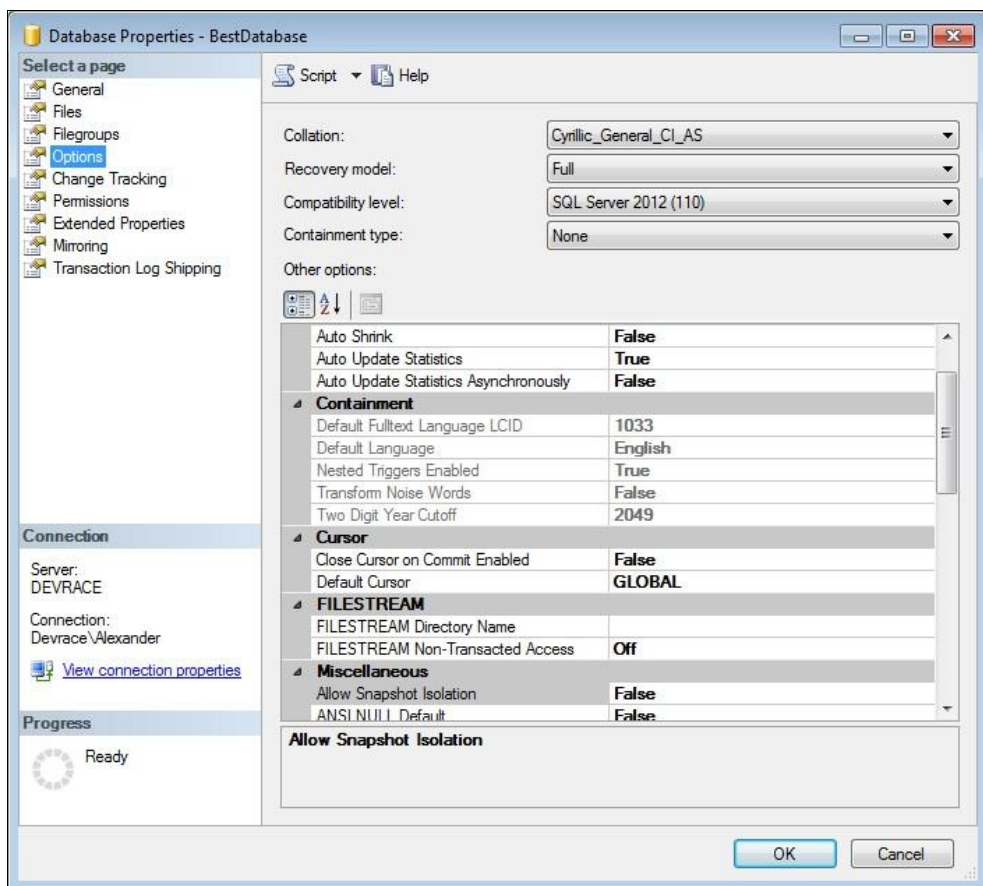


Рис. 3.14. Свойства базы данных BestDatabase. Вкладка **Options**

Теперь посмотрим на характеристики базы данных MultyGroup, которая содержит не одну, а две файловые группы.

Закройте диалоговое окно просмотра свойств базы данных BestDatabase, щелкнув мышью по кнопке **Cancel** в любой вкладке этого окна. Щелкните правой кнопкой мыши по имени базы данных MultyGroup в **Object Explorer** и в появившемся контекстном меню выберите строку **Properties**. Откроется окно просмотра свойств этой базы данных.

Выберите вкладку **Files** для просмотра файлов базы данных. Список файлов показан на рис. 3.15.

Database files:						
Logical Name	File Type	Filegroup	Initial Size (MB)	Autogrowth / Maxsize	Path	File Name
MultyGroup1	Rows Data	PRIMARY	5	By 1 MB, Unlimited	...	D:\MultyGroup\MultyGroup1.mdf
MultyGroup3	Rows Data	MultyGroup2	1	By 1 MB, Unlimited	...	D:\MultyGroup\MultyGroup3.ndf
MultyGroup4	Rows Data	MultyGroup2	1	By 1 MB, Unlimited	...	D:\MultyGroup\MultyGroup4.ndf
MultyGroup2	Rows Data	PRIMARY	1	By 1 MB, Unlimited	...	D:\MultyGroup\MultyGroup2.ndf
MultyGroupLog1	Log	Not Applicable	1	By 10 percent, Limited to 209...	...	D:\MultyGroup\MultyGroupLog1.ldf
MultyGroupLog2	Log	Not Applicable	1	By 10 percent, Limited to 209...	...	D:\MultyGroup\MultyGroupLog2.ldf

Рис. 3.15. Список файлов базы данных MultyGroup

Главное, что здесь можно увидеть интересного в отличие от списка файлов базы данных BestDatabase, это распределение файлов по файловым группам. Мы видим, что файлы данных MultyGroup1 и MultyGroup2 принадлежат первичной файловой группе PRIMARY, а вторичной файловой группе MultyGroup2 принадлежат файлы данных MultyGroup3 и MultyGroup4.

Теперь на панели **Select a page** выберите вкладку **Filegroups** (файловые группы). Данные на этой вкладке весьма скромненькие. Впрочем, на большее и рассчитывать-то не стоило. Что особенного можно сказать про файловые группы?

В верхней части окна указаны обе файловые группы этой базы данных и представлено количество файлов данных, входящих в каждую группу. Для файловой группы PRIMARY стоит отметка в столбце Default, это означает, что она является первичной файловой группой, файловой группой по умолчанию.

К вкладкам свойств базы данных мы будем неоднократно возвращаться, в том числе и в этой главе, когда станем изменять характеристики существующих баз данных.

Сейчас же рассмотрим диалоговые средства Management Studio, используемые для создания новых баз данных. Закройте диалоговое окно просмотра свойств базы данных, щелкнув по кнопке **Cancel**.

Замечания по использованию Management Studio

В Management Studio есть очень удобная возможность в окне **New Query** (создать запрос) вводить произвольное количество операторов, а для выполнения одного оператора или группы операторов нужно выделить необходимую группу строк и нажать клавишу <F5> или комбинацию клавиш <Ctrl>+<E>.

Для упрощения получения нужных результатов в подходящем виде в Management Studio есть еще ряд полезных возможностей. Мы выводили все результаты выполнения запросов в окне **New Query** в табличном виде или в так называемую сетку (Grid). Существует возможность выводить результаты в текстовом виде, практически так же, как они отображаются в утилите `sqlcmd` при использовании командной строки (PowerShell). Для этого перед выполнением группы операторов нужно в меню выбрать **Query | Results To | Results to Text** или нажать клавиши `<Ctrl>+<T>`.

Чтобы вернуться к более привычной форме отображения результатов в табличном виде, нужно выбрать в меню **Query | Results To | Results to Grid** или нажать клавиши `<Ctrl>+<D>`.

Кроме этих возможностей в Management Studio существует еще множество дополнительных настроек. Для получения доступа к многочисленным настройкам выберите в меню **Query | Query Options** или щелкните мышью по кнопке **Query Options** на инструментальной панели. Появится окно **Query Options**, в котором на нескольких вкладках можно установить большое количество характеристик, используемых при выполнении программы. Например, на вкладке **Results** (результаты) можно указать необходимость включения текста запроса в результат его выполнения, можно задать отображение результата выполнения запроса в отдельной таблице. Вместо оператора `GO`, разделяющего выполняемые фрагменты пакета, на вкладке **General** можно указать другой оператор, просто введя его текст в соответствующее поле, чего, конечно же, никак нельзя порекомендовать.

Можно указать, что в окне запроса должны отображаться номера строк. Для этого нужно выбрать в меню **Tools | Options**. Появится диалоговое окно задания режимов. В левой части окна нужно раскрыть элемент **Transact-SQL** и щелкнуть мышью по элементу **General**. В правой части окна в разделе **Display** нужно установить флажок **Line numbers** и щелкнуть по кнопке **OK**.

После этой установки в левой части окна запроса будут выводиться номера строк. Это полезная возможность, особенно тогда, когда вы отлаживаете достаточно "длинный" запрос, содержащий большое количество строк, например, добавление многих строк в таблицы базы данных. В случае ошибок система выдает соответствующее сообщение с указанием номера строки и номера позиции, где была обнаружена ошибка.

Если вы вводите неправильную конструкцию, то система подчеркивает неверные символы красной волнистой линией. Подведя к ошибочному тексту курсор мыши, вы получите краткую подсказку, что именно не так вы сделали в операторе. Правда, бывают случаи, когда подчеркиваются и совершенно верные тексты. Но это случается достаточно редко.

3.4.2. Создание базы данных с использованием диалоговых средств Management Studio

Базу данных можно создавать не только с использованием оператора `Transact-SQL CREATE DATABASE`. В Management Studio существуют диалоговые средства, позволяющие задать все необходимые характеристики создаваемой базы данных.

Щелкните правой кнопкой мыши в **Object Explorer** по строке **Databases** и в появившемся контекстном меню выберите элемент **New Database** (можно также щелкнуть правой кнопкой мыши по имени *любой* пользовательской базы данных и выбрать строку **New Database**). Появится окно создания новой базы данных **New Database** (рис. 3.16), где можно создавать базу данных, ее файлы со всеми необходимыми характеристиками. Текущей будет вкладка **General**.

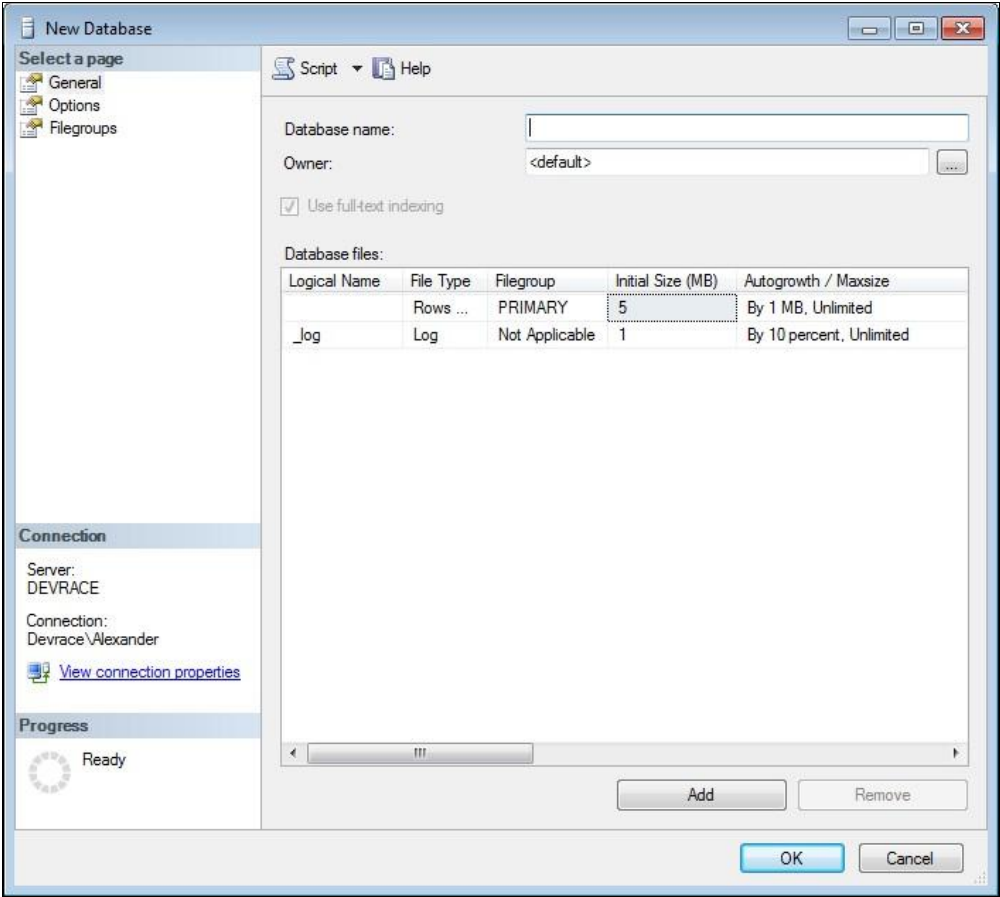


Рис. 3.16. Окно создания новой базы данных **New Database**

Вначале зададим имя базы данных. Пусть это будет база данных с простым и понятным именем **NewDatabase**. Введите это имя в поле **Database name** в верхней части окна. В процессе ввода имени в строках столбца **Logical Name** появляются соответствующие имена **NewDatabase** (файл данных) и **NewDatabase_log** (журнал транзакций). Оставим все как есть, только к имени файла данных добавим еще суффикс **_dat**.

В столбце **Path** (пути к файлам базы данных) программа предлагает нам некоторый вариант размещения файлов создаваемой базы данных. В правой части строки в этом поле присутствует кнопка с многоточием. Для выбора другого размещения

файлов щелкните по этой кнопке. Откроется окно выбора папки — **Locate Folder** (рис. 3.17). Выберите по очереди для каждого из файлов тот же самый каталог, что мы использовали для базы данных BestDatabase: на диске D:\BestDatabase.

Щелкните в этом окне по кнопке **ОК**.



Рис. 3.17. Выбор папки для размещения файла базы данных

Каталог для размещения файлов можно задать и путем ввода в поля столбца **Path** нужного пути к файлам, просто набрав с клавиатуры текст D:\BestDatabase.

В столбце **File Name** введите имена файлов, соответственно для файла данных и файла журнала транзакций: NewDatabase.mdf и NewDatabase.ldf. В столбце **Logical Name** введите логические имена файлов: NewDatabase_dat и NewDatabase_log.

Список файлов будет выглядеть так, как на рис. 3.18.

Database files:						
Logical Name	File Type	Filegroup	Initial ...	Autogrowth / Maxsize	Path	File Name
NewDatabase_dat	Rows Data	PRIMARY	5	By 1 MB, Unlimited	D:\BestDatabase	NewDatabase.mdf
NewDatabase_log	Log	Not Applicable	1	By 10 percent, Unlimited	D:\BestDatabase	NewDatabase.ldf

Рис. 3.18. Файлы создаваемой базы данных

Для создания этой базы данных нужно щелкнуть по кнопке **ОК**, однако давайте пока повременим с этим и рассмотрим еще возможности добавления вторичных файловых групп и новых файлов.

Если для создаваемой базы данных вам нужно задать несколько файлов данных и/или несколько файлов журнала транзакций, то вы можете щелкнуть мышью по кнопке **Add** в нижней части окна и добавить любое количество файлов данных или файлов журналов транзакций.

Для этой базы данных создадим еще один файл данных с именем NewDatabase2. Щелкните по кнопке **Add**. В окне появится новая строка с заданными значениями некоторых характеристик. В качестве логического имени нового файла введите NewDatabase2, выберите или введите с клавиатуры тот же путь к файлу (**Path**), что и для других файлов этой базы данных, задайте для него и имя физического файла NewDatabase2.ndf.

Тип файла (**File Type**) по умолчанию устанавливается как файл данных (Rows Data). Если вы собираетесь создавать файл журнала транзакций, то нужно в поле **File Type** щелкнуть мышью справа от значения этого поля по кнопке со стрелкой вниз.

Появится раскрывающийся список, в котором вы в этом случае должны были бы выбрать значение `Log`. Сейчас оставим для типа файла значение по умолчанию.

Аналогичным образом вы можете выбрать файловую группу, которой будет принадлежать создаваемый файл — разумеется, только если вы создаете файл данных. Для этого нужно щелкнуть по кнопке со стрелкой вниз в правой части поля **Filegroup** и из раскрывающегося списка выбрать нужную файловую группу. Поскольку никаких вторичных файловых групп мы пока не создавали, список будет содержать только две строки: `PRIMARY` и `<new filegroup>` (новая файловая группа).

Создадим новую файловую группу этим способом. Другой путь создания новой файловой группы мы используем при внесении изменения в эту базу данных.

Выберите в этом раскрывающемся списке строку `<new filegroup>`. Появится окно создания новой файловой группы **New Filegroup**. Введите в поле **Name** имя файловой группы, которую также без затей и назовем: `NewGroup` (рис. 3.19).

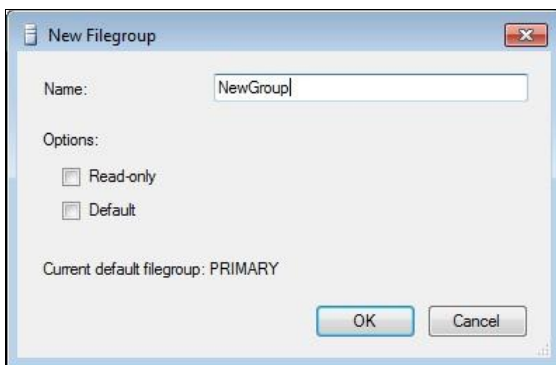


Рис. 3.19. Создание новой файловой группы

Щелкните по кнопке **OK**. Будет создана новая файловая группа, которой будет принадлежать и вновь создаваемый файл `NewDatabase2`.

Установите для файла начальный размер (столбец **Initial Size**) в 2 Мбайта, введя с клавиатуры число 2 в соответствующей строке этого столбца или установив значение, используя кнопку с изображением стрелки вверх (для увеличения значения на единицу) или стрелки вниз (для уменьшения значения на единицу), как это показано на рис. 3.20.



Рис. 3.20. Задание начального размера файла

Для задания характеристик роста размера файла щелкните мышью по кнопке с многоточием в правой части строки столбца **Autogrowth**. Появится диалоговое окно **Change Autogrowth**. В этом окне установите следующие характеристики.

- ♦ В группе переключателей **File Growth** (увеличение размера файла) выберите **In Megabytes** (в мегабайтах), чтобы приращение указывалось в мегабайтах, и за-

- дайте величину приращения 2, введя это значение вручную или с использованием кнопок с изображением стрелки вверх или стрелки вниз в правой части соответствующего счетчика.
- ♦ В группе переключателей **Maximum File Size** (максимальный размер файла) выберите **Limited to (MB)** (ограничение на рост размера файла). Укажите максимальный размер файла в 20 Мбайт, введя значение в счетчик вручную или используя кнопки с изображением стрелки вверх и стрелки вниз.
- Выбранные значения показаны на рис. 3.21.

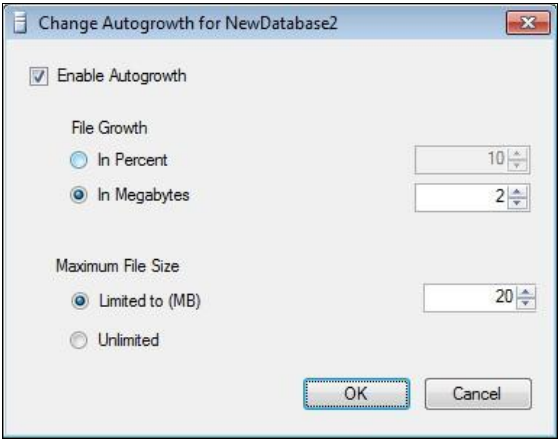


Рис. 3.21. Изменение характеристик увеличения размера файла

Щелкните мышью по кнопке **ОК**. Будут сформированы все заданные значения для второго файла данных. Список файлов создаваемой базы данных теперь будет выглядеть, как показано на рис. 3.22.

Для завершения создания базы данных щелкните по кнопке **ОК**.

Database files:							
Logical Name	File Type	Filegroup	Initial ...	Autogrowth / Maxsize	Path	File Name	
NewDatabase_dat	Rows Data	PRIMARY	5	By 1 MB, Unlimited	D:\BestDatabase	NewDatabase.mdf	
NewDatabase_log	Log	Not Applicable	1	By 10 percent, Unlimited	D:\BestDatabase	NewDatabase.ldf	
NewDatabase2	Rows Data	NewGroup	2	By 2 MB, Limited to 20 MB	D:\BestDatabase	NewDatabase2.ndf	

Рис. 3.22. Новый список файлов базы данных

3.5. Изменение базы данных

Изменять характеристики существующей в экземпляре сервера пользовательской базы данных можно оператором Transact-SQL `ALTER DATABASE` или при использовании диалоговых средств Management Studio. Рассмотрим оба варианта.

Следует помнить, что вносить изменения можно в ту базу данных, с которой в этот момент не соединены другие пользователи.

3.5.1. Изменение базы данных в языке Transact-SQL

Для изменения существующей пользовательской базы данных в языке Transact-SQL используется оператор `ALTER DATABASE`. Его синтаксис представлен в листинге 3.5 и соответствующем R-графе (граф 3.9).

Листинг 3.5. Синтаксис оператора `ALTER DATABASE`

```
ALTER DATABASE { <имя базы данных> | CURRENT }
{ MODIFY NAME = <новое имя базы данных>
| COLLATE <порядок сортировки>
| <характеристики файлов>
| <характеристики файловых групп>
| <характеристики базы данных>
};
```



Граф 3.9. Синтаксис оператора `ALTER DATABASE`

Изменения будут выполняться либо для базы данных, указанной в операторе по имени, либо для текущей базы данных, заданной в последнем операторе `USE`, в случае указания ключевого слова `CURRENT`.

Как видно из синтаксиса, за одно выполнение оператора можно либо переименовать базу данных, либо изменить порядок сортировки, либо изменить характеристики файлов, файловых групп или всей базы данных. Для выполнения нескольких действий нужно использовать соответствующее количество операторов `ALTER DATABASE`.

3.5.1.1. Изменение имени базы данных

Самое простое действие — изменение имени базы данных. Для этого используется предложение `MODIFY NAME`, где указывается новое имя базы данных. Это имя должно быть, естественно, уникальным в данном экземпляре сервера.

Измените имя базы данных `BestDatabase` на `NewBest`. Выполните операторы из примера 3.22 в утилите `sqlcmd` или в окне выполнения запросов `Management Studio`.

Пример 3.22. Изменение имени базы данных BestDatabase

```
USE master;
GO
ALTER DATABASE BestDatabase
    MODIFY NAME = NewBest;
GO
```

В строке утилиты `sqlcmd` или на вкладке **Messages** компонента Management Studio (в зависимости от того, каким средством вы сейчас пользовались) появится сообщение об изменении имени:

The database name 'NewBest' has been set.

Отобразите список имен и порядков сортировки баз данных (пример 3.23).

Пример 3.23. Отображение списка баз данных и порядков сортировки

```
SELECT CAST(name AS CHAR(20)) AS 'NAME',
       CAST(collation_name AS CHAR(23)) AS 'COLLATION'
FROM sys.databases;
GO
```

Видно, что имя базы данных было изменено. Верните нашей базе данных ее родное имя **BestDatabase**, выполнив операторы:

```
USE master;
GO
ALTER DATABASE NewBest
    MODIFY NAME = BestDatabase;
GO
```

3.5.1.2. Изменение порядка сортировки

Для базы данных также весьма просто можно изменить порядок сортировки в операторе `ALTER DATABASE`. Выполните пакет из примера 3.24 (одна моя знакомая программистка уехала во Францию и перед отъездом проявила живейший интерес к порядку сортировки из данного примера).

Пример 3.24. Изменение порядка сортировки базы данных BestDatabase

```
USE master;
GO
ALTER DATABASE BestDatabase
    COLLATE French_CI_AI;
GO
```

В текущем состоянии базы данных **BestDatabase** все пройдет нормально, однако если вы создавали в этой базе данных таблицы (что мы выполним в одной из сле-

дующих глав), то можете получить сообщения об ошибке. В частности, изменение порядка сортировки невозможно, если отдельные строковые столбцы, для которых не задана сортировка, отличная от сортировки по умолчанию, присутствуют в ограничениях CHECK или включены в состав вычисляемых столбцов.

Если изменение существующего порядка сортировки по умолчанию возможно для базы данных, то новое значение будет влиять только лишь на вновь создаваемые столбцы существующих или новых таблиц. Чтобы сохранить существующие данные, потребуется выполнить действия по загрузке/выгрузке данных. Если изменения касаются столбцов, входящих в состав индексов, то нужно будет и перестроить соответствующие индексы.

Отобразите опять список баз данных (см. пример 3.23) и убедитесь, что порядок сортировки нашей базы данных изменился.

При помощи оператора `ALTER DATABASE` верните порядок сортировки базы данных `BestDatabase` к значению по умолчанию, принятому в текущем экземпляре сервера — `Cyrillic_General_CI_AS`.

Пример 3.25. Обратное изменение порядка сортировки базы данных `BestDatabase`

```
USE master;
GO
ALTER DATABASE BestDatabase
    COLLATE Cyrillic_General_CI_AS;
GO
```

Чтобы просмотреть все существующие в текущем экземпляре сервера базы данных порядки сортировки, используйте функцию `fn_helpcollations()`. Для каждого порядка сортировки функция возвращает два столбца: `Name`, содержащий имя порядка сортировки, и `Description`, в котором хранится текстовое описание.

Для получения списка порядков сортировки выполните следующий оператор:

```
SELECT Name, Description FROM fn_helpcollations();
```

Вы получите более двух тысяч строк. Чтобы выделить из списка только те порядки сортировки, которые связаны с кириллицей, добавьте в оператор `SELECT` предложение `WHERE` (пример 3.26).

Пример 3.26. Отображение списка порядков сортировки системы

```
USE master;
GO
SELECT Name, Description FROM fn_helpcollations()
    WHERE name LIKE 'Cyrillic%';
GO
```

Здесь в результирующий набор данных попадут лишь те порядки сортировки, имена у которых начинаются с символов `'Cyrillic'`. На момент написания данной главы это будут 52 строки:

```
Cyrillic_General_BIN  
Cyrillic_General_BIN2  
Cyrillic_General_CI_AI  
Cyrillic_General_CI_AI_WS  
Cyrillic_General_CI_AI_KS  
Cyrillic_General_CI_AI_KS_WS  
Cyrillic_General_CI_AS  
Cyrillic_General_CI_AS_WS  
Cyrillic_General_CI_AS_KS  
Cyrillic_General_CI_AS_KS_WS  
...
```

В поле `Description` обычным текстом (разумеется, на английском языке) описываются характеристики порядка сортировки, в первую очередь, чувствительность к регистру. Подробнее возможные конструкции в предложении `WHERE` оператора `SELECT` мы будем рассматривать позже. В следующей главе при рассмотрении строковых типов данных мы исследуем их чувствительность к регистру.

Чтобы завершить обсуждение вопросов сортировки, давайте еще кратко рассмотрим функцию `COLLATIONPROPERTY()`. Ее синтаксис следующий:

```
COLLATIONPROPERTY(<имя порядка сортировки>, <свойство>)
```

Первый передаваемый функции параметр — имя порядка сортировки. Второй параметр — интересующее нас свойство. Функция возвращает значение требуемого свойства указанного порядка сортировки. Если функции в качестве параметра будет передано неверное имя порядка сортировки или имя свойства, не предусмотренное в функции, то она вернет значение `NULL`.

Именами свойств являются следующие:

- ◆ `CodePage` — кодовая страница порядка сортировки;
- ◆ `LCID` — код языка в Windows;
- ◆ `ComparisonStyle` — стиль сравнения Windows;
- ◆ `Version` — версия порядка сортировки. Число 0, 1 или 2.

Например, чтобы получить значение кодовой страницы и кода языка для порядка сортировки `Cyrillic_General_CI_AS`, нужно выполнить следующий оператор `SELECT` (пример 3.27).

Пример 3.27. Отображение характеристик порядка сортировки

```
USE master;  
GO  
SELECT COLLATIONPROPERTY('Cyrillic_General_CI_AS', 'CodePage'),  
       COLLATIONPROPERTY('Cyrillic_General_CI_AS', 'LCID');  
GO
```

Результатом будут два значения: 1251 (кодовая страница) и 1049 (код языка).


```
[ , MAXSIZE = { <целое2> [ KB | MB | GB | TB ] | UNLIMITED } ]  
[ , FILEGROWTH = <целое3> [ KB | MB | GB | TB | % ] ]  
)
```

Поскольку мы с вами прекрасно умеем создавать новую базу со всеми ее файлами, то здесь нам все понятно.

Удаление существующего файла

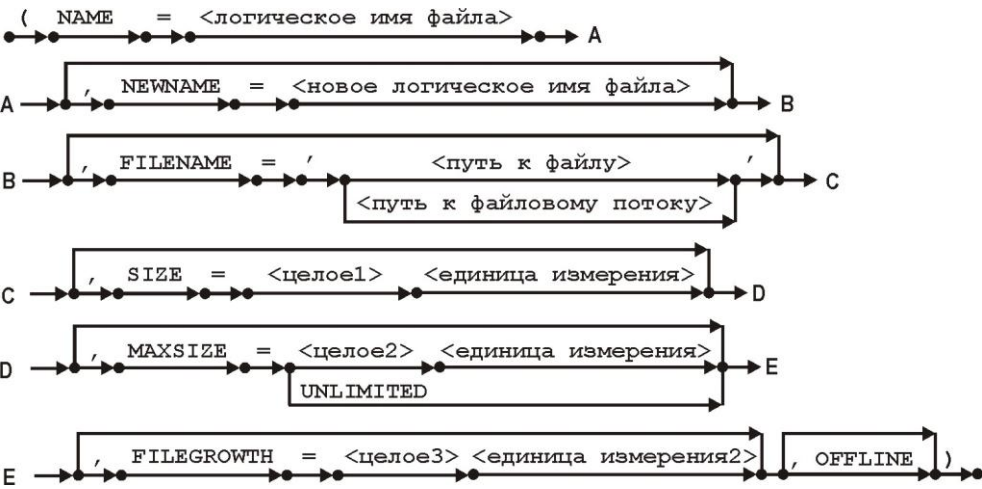
Чтобы удалить существующий в базе файл данных или файл журнала транзакций, используется предложение REMOVE FILE, в котором нужно указать логическое имя файла. Нельзя удалить единственный в базе данных файл данных или единственный файл журнала транзакций. Нельзя также удалить файл, содержащий данные.

Изменение характеристик файла

Для переименования или изменения характеристик существующего файла данных или файла журнала транзакций используется предложение MODIFY FILE. Синтаксическая конструкция "изменяемый файл" в этом предложении очень похожа на обычную спецификацию файла (см. листинг 3.7 и граф 3.11).

Листинг 3.7. Синтаксис задания изменяемого файла

```
<изменяемый файл> ::=  
( NAME = <логическое имя файла>  
  [ , NEWNAME = <новое логическое имя файла> ]  
  [ , FILENAME = { '<путь к файлу>' | '<путь к файловому потоку>' } ]  
  [ , SIZE = <целое1> [ KB | MB | GB | TB ] ]  
  [ , MAXSIZE = { <целое2> [ KB | MB | GB | TB ] | UNLIMITED } ]  
  [ , FILEGROWTH = <целое3> [ KB | MB | GB | TB | % ] ]  
  [ , OFFLINE ]  
)
```



Граф 3.11. Синтаксис задания изменяемого файла

ЗАМЕЧАНИЕ

Последнюю опцию в этом фрагменте синтаксиса, `OFFLINE`, следует использовать только в случае разрушения файла. Перевести обратно в активное состояние такой файл можно лишь при восстановлении файла из резервной копии.

`NAME` задает логическое имя изменяемого файла.

Вариант `NEWNAME` позволяет задать новое логическое имя для файла. Новое имя должно быть уникальным в этой базе данных.

Здесь также можно изменить значение начального размера файла, максимального размера, а также величины приращения.

Можно переместить файл в другое место на внешнем носителе, задав новый путь в варианте `FILENAME`. Можно переименовать файл в том же самом каталоге или переместив его в любой другой каталог и на другой внешний носитель. Однако не так все просто. Чтобы переименовать файл данных или файл журнала транзакций и/или переместить его в другой каталог или на другой носитель, нужно выполнить ряд действий, и не только с использованием операторов Transact-SQL.

Рассмотрим на примере нашей базы данных `BestDatabase` переименование файла данных, скажем, в `WinnerNew.mdf`. Вначале нужно будет перевести базу данных в неактивное состояние `OFFLINE`. Затем нужно выполнить оператор `ALTER DATABASE` для переименования файла, оставив его в том же каталоге. Это можно сделать в том же пакете. Выполните следующие операторы (пример 3.28).

Пример 3.28. Переименование файла данных базы данных `BestDatabase`

```
USE master;  
GO  
ALTER DATABASE BestDatabase SET OFFLINE;  
ALTER DATABASE BestDatabase  
MODIFY FILE (NAME = BestDatabase_dat,  
    FILENAME = 'D:\BestDatabase\WinnerNew.mdf');  
GO
```

Мы получим следующее информационное сообщение системы:

```
The file "BestDatabase_dat" has been modified in the system catalog.  
The new path will be used the next time the database is started.
```

Имя нашего файла было изменено в системном каталоге, а новый путь будет использован только после "перезапуска" нашей базы данных, т. е. после перевода ее опять в активное состояние.

Оператор `ALTER DATABASE` переименовывает файл базы данных *только в каталоге* сервера базы данных, но не на внешнем носителе. Средствами операционной системы, например, при помощи программы Компьютер переименуйте файл. После этого базу данных можно перевести в оперативное состояние `ONLINE`, выполнив следующий оператор:

```
ALTER DATABASE BestDatabase SET ONLINE;
```

Аналогичным образом выполняется и перемещение файла на другой носитель или в другой каталог. Вначале база данных переводится в состояние `OFFLINE`, затем при помощи оператора `ALTER DATABASE` изменяется путь к файлу. Любыми доступными средствами вы переписываете файл на нужный новый носитель в указанный каталог. После этого переводите базу данных в состояние `ONLINE`.

За одно выполнение оператора `ALTER DATABASE` можно изменить имя или положение только одного файла базы данных. Если, например, вы хотите переименовать и файл данных, и файл журнала транзакций, то вам нужно дважды выполнить оператор `ALTER DATABASE` (не считая оператор перевода базы данных в неактивное состояние). Выполните пакет операторов в примере 3.29.

Пример 3.29. Переименование обоих файлов базы данных BestDatabase

```
USE master;
GO
ALTER DATABASE BestDatabase SET OFFLINE;
ALTER DATABASE BestDatabase
MODIFY FILE (NAME = BestDatabase_dat,
    FILENAME = 'D:\BestDatabase\WinnerNew.mdf');
ALTER DATABASE BestDatabase
MODIFY FILE (NAME = BestDatabase_log,
    FILENAME = 'D:\BestDatabase\WinnerNew.ldf');
GO
```

Не забудьте после переименования файлов на внешнем носителе вернуть базу данных в состояние `ONLINE`.

Теперь к нашей базе данных добавим еще два файла — файл данных и файл журнала транзакций. Выполните операторы, записанные в примере 3.30.

Пример 3.30. Добавление файлов к базе данных

```
USE master;
GO
ALTER DATABASE BestDatabase
ADD FILE (NAME = BestDatabase_dat2,
    FILENAME = 'D:\BestDatabase\Winner2.mdf',
    SIZE = 5 MB,
    MAXSIZE = UNLIMITED,
    FILEGROWTH = 1 MB);
ALTER DATABASE BestDatabase
ADD LOG FILE (NAME = BestDatabase_log2,
    FILENAME = 'D:\BestDatabase\Winner2.ldf',
    SIZE = 2 MB,
    MAXSIZE = 30 MB,
    FILEGROWTH = 1 MB);
GO
```

Проверьте результат выполнения пакета. Вы можете увидеть, что к базе данных добавлены два новых файла, а на внешнем носителе созданы файлы с соответствующими характеристиками.

Рассмотрим еще пример, в котором изменяется начальный и максимальный размер первого файла данных. Оператор представлен в листинге примера 3.31.

Пример 3.31. Изменение размера файла данных

```
USE master;
GO
ALTER DATABASE BestDatabase
MODIFY FILE (NAME = BestDatabase_dat,
    SIZE = 100 MB,
    MAXSIZE = 1000 MB);
GO
```

Новое значение начального размера файла не должно быть равным или меньшим, чем текущее значение. Что любопытно — система не проверяет, является ли максимальное значение размера большим, чем начальное.

Выполним удаление файлов базы данных. Попробуйте удалить первичный файл данных (пример 3.32).

Пример 3.32. Попытка удаления первичного файла данных

```
USE master;
GO
ALTER DATABASE BestDatabase
    REMOVE FILE BestDatabase_dat;
GO
```

Вы получите следующее сообщение:

```
The primary data or log file cannot be removed from a database.
```

Нам напоминают, что из базы данных нельзя удалить первичный файл данных или файл журнала транзакций.

А вот вторичные файлы, если они не заполнены данными, можно легко удалить (пример 3.33).

Пример 3.33. Удаление файла данных

```
USE master;
GO
ALTER DATABASE BestDatabase
    REMOVE FILE BestDatabase_dat2;
GO
```

Эта операция проходит нормально. Система удаляет описание логического файла из системного каталога и также удаляет физический файл с внешнего носителя.

Подведем маленький итог по работе с файлами базы данных.

Когда мы создаем базу данных или добавляем в существующую базу новые файлы, система не только добавляет сведения о файлах в системный каталог, но и физически создает файлы с заданными характеристиками на внешних носителях в указанных каталогах. Каталоги в пути к файлам должны уже существовать на диске.

Аналогично, при удалении какого-либо файла из базы система корректирует системный каталог и физически удаляет с внешнего носителя соответствующий файл.

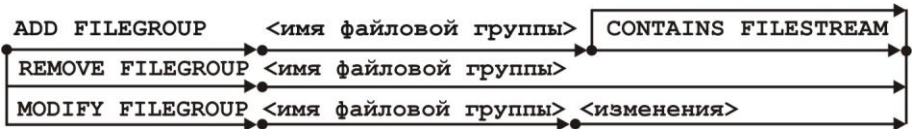
Однако если вы изменяете имя любого файла базы данных или перемещаете его в другой каталог или на другой внешний носитель, вам, во-первых, нужно перевести базу данных в состояние OFFLINE и, во-вторых, требуется физически средствами операционной системы изменить имя или переместить файл в другое место. После этого нужно перевести базу данных в состояние ONLINE.

3.5.1.4. Изменение файловых групп

В синтаксисе оператора ALTER DATABASE (см. ранее листинг 3.5) указана конструкция "характеристики файловых групп". Синтаксис, несколько упрощенный, этой конструкции представлен в листинге 3.8 и в R-графах (графы 3.12 и 3.13).

Листинг 3.8. Синтаксис изменения файловых групп

```
<характеристики файловых групп> ::=
{
    ADD FILEGROUP <имя файловой группы> [ CONTAINS FILESTREAM ]
  | REMOVE FILEGROUP <имя файловой группы>
  | MODIFY FILEGROUP <имя файловой группы>
  {
      { READ_ONLY | READ_WRITE }
      | DEFAULT
      | NAME = <имя новой файловой группы>
  }
}
```



Граф 3.12. Синтаксис изменения файловых групп

Name	Files	Read-Only	Default
PRIMARY	2		<input checked="" type="checkbox"/>
MultyGroup0	2	<input type="checkbox"/>	<input type="checkbox"/>
MultyGroup2	0	<input type="checkbox"/>	<input type="checkbox"/>

Рис. 3.23. Измененный список файловых групп

3.5.1.5. Изменение других характеристик базы данных

В описании синтаксиса оператора `ALTER DATABASE` (см. листинг 3.5) последней строкой присутствует нетерминальный символ "характеристики базы данных". Эти характеристики задаются предложением `SET` в операторе `ALTER DATABASE` (листинг 3.9).

Листинг 3.9. Синтаксис изменения характеристик базы данных

```
ALTER DATABASE { <имя базы данных> | CURRENT }
SET <характеристика> [, <характеристика>]...;
```

Вкратце перечислим существующие характеристики. Достаточно подробно они описаны в *приложении 4*.

- ◆ `AUTO_CLOSE { ON | OFF }` — задает автоматическое закрытие базы данных после отключения от нее последнего пользователя.
- ◆ `AUTO_CREATE_STATISTICS { ON | OFF }` — задает или отключает автоматическое создание статистики по базе данных, требуемой для оптимизации запроса.
- ◆ `AUTO_UPDATE_STATISTICS { ON | OFF }` — задает или отключает автоматическое обновление статистики по базе данных, требуемой для оптимизации запроса.
- ◆ `AUTO_UPDATE_STATISTICS_ASYNC { ON | OFF }` — влияет на выполнение запросов, которые требуют обновления статистики по базе данных.
- ◆ `AUTO_SHRINK { ON | OFF }` — задает или отключает режим автоматического сжатия файлов базы данных.
- ◆ Состояние базы данных: `{ ONLINE | OFFLINE | EMERGENCY }`.
- ◆ Вид базы данных: обычная, частично автономная:
`CONTAINMENT = { NONE | PARTIAL }.`
- ◆ Возможность изменения данных базы данных: `{ READ_ONLY | READ_WRITE }`.
- ◆ Допустимое количество подключаемых к базе данных пользователей:
`{ SINGLE_USER | RESTRICTED_USER | MULTI_USER }`.
- ◆ `CURSOR_CLOSE_ON_COMMIT { ON | OFF }` — задает автоматическое закрытие курсора при подтверждении транзакции.

- ◆ `CURSOR_DEFAULT { GLOBAL | LOCAL }` — задает область действия курсора.
- ◆ `RECOVERY { FULL | BULK_LOGGED | SIMPLE }` — определяет стратегии создания резервных копий и восстановления поврежденной базы данных.
- ◆ `PAGE_VERIFY { CHECKSUM | TORN_PAGE_DETECTION | NONE }` — задает способ обнаружения поврежденных страниц базы данных.
- ◆ `ANSI_NULL_DEFAULT { ON | OFF }` — определяет значение по умолчанию для столбцов таблиц: `NULL` или `NOT NULL`.
- ◆ `ANSI_NULLS { ON | OFF }` — задает соответствие стандарту ANSI результат сравнения любых значений со значением `NULL`.
- ◆ `ANSI_PADDING { ON | OFF }` — влияет на добавление конечных пробелов в столбцы типов данных `VARCHAR` и `NVARCHAR`, а также на конечные нули в двоичных значениях `VARBINARY`.
- ◆ `ANSI_WARNINGS { ON | OFF }` — влияет на появление предупреждающих сообщений или сообщений об ошибке при появлении значения `NULL` в агрегатных функциях или при делении на ноль.
- ◆ `ARITHABORT { ON | OFF }` — определяет реакцию системы на арифметическое переполнение или при делении на ноль: прекращение выполнения запроса или выдача сообщения и продолжение выполнения запроса.
- ◆ `COMPATIBILITY_LEVEL = { 80 | 90 | 100 | 110 }` — уровень совместимости с предыдущими версиями SQL Server.
- ◆ `CONCAT_NULL_YIELDS_NULL { ON | OFF }` — определяет результат конкатенации двух строковых данных, когда одно из строковых значений имеет значение `NULL`.
- ◆ `DATE_CORRELATION_OPTIMIZATION { ON | OFF }` — определяет поддержание статистики между таблицами, связанными ограничением `FOREIGN KEY` и содержащими столбцы типа данных `datetime`.
- ◆ `NUMERIC_ROUNDABORT { ON | OFF }` — задает возможность появления ошибки при потере точности в процессе вычисления числового значения.
- ◆ `PARAMETERIZATION { SIMPLE | FORCED }` — условие выполнения параметризации запросов.
- ◆ `QUOTED_IDENTIFIER { ON | OFF }` — определяет допустимость использования кавычек при задании идентификаторов с разделителями.
- ◆ `RECURSIVE_TRIGGERS { ON | OFF }` — определяет допустимость срабатывания рекурсивных триггеров.
- ◆ `DB_CHAINING { ON | OFF }` — задает, может ли база данных находиться в межбазовой цепочке владения или быть источником в такой цепочке.
- ◆ `TRUSTWORTHY { ON | OFF }` — определяет, могут ли модули базы данных получать доступ к ресурсам вне базы данных.
- ◆ `ALLOW_SNAPSHOT_ISOLATION { ON | OFF }` — определяет допустимость использования уровня изоляции транзакции `SNAPSHOT`.

◆ `READ_COMMITTED_SNAPSHOT { ON | OFF }` — определяет поведение транзакции с уровнем изоляции `READ COMMITTED`.

Давайте из праздного любопытства изменим значения первых пяти перечисленных характеристик базы данных `BestDatabase` (пример 3.35). Мы установим для этих характеристик значения, отличные от значений по умолчанию, присваиваемых базе данных при ее создании. Рассматривайте этот пример только как исследование. Совсем не обязательно для реальной базы данных, например, отменять создание статистических данных, которые позволяют сильно повысить производительность системы при обработке запросов.

Пример 3.35. Изменение некоторых характеристик базы данных `BestDatabase`

```
USE master;
GO
ALTER DATABASE BestDatabase
SET AUTO_CLOSE ON,
    AUTO_CREATE_STATISTICS OFF,
    AUTO_UPDATE_STATISTICS OFF,
    AUTO_UPDATE_STATISTICS_ASYNC ON,
    AUTO_SHRINK ON;
GO
```

3.5.2. Изменение базы данных диалоговыми средствами Management Studio

В Management Studio можно изменять несколько меньше характеристик базы данных, чем при использовании оператора `ALTER DATABASE`.

3.5.2.1. Изменение имени базы данных

Базу данных очень легко переименовать. Для этого нужно щелкнуть правой кнопкой по имени базы данных и в появившемся контекстном меню выбрать пункт **Rename** (переименовать). Имя базы данных станет доступным для изменения непосредственно в панели **Object Explorer**. Такой же эффект можно получить, если аккуратно щелкнуть мышью по самому имени базы данных в **Object Explorer**.

Чтобы отобразить окно свойств базы данных, нужно в **Object Explorer** щелкнуть правой кнопкой мыши по имени базы данных и в контекстном меню выбрать элемент **Properties**. Появится окно свойств.

3.5.2.2. Изменение файлов базы данных

Если в окне свойств базы данных `BestDatabase` выбрать вкладку **Files**, то появится список файлов этой базы, как это показано на рис. 3.24.

На этой вкладке мы можем не только изменять характеристики существующих файлов, но также добавлять и удалять файлы.



Logical Name	File Type	Filegroup	Initi...	Autogrowth / Maxsize	Path	File Name
BestDatabase_dat	Rows ...	PRIMARY	5	By 1 MB, Unlimited		D:\BestDatabase Winner.mdf
BestDatabase_log	Log	Not Applicable	1	By 10 percent, Limited to 2097152 MB		D:\BestDatabase Winner.ldf

Рис. 3.24. Характеристики файлов базы данных BestDatabase

Здесь можно изменить логическое имя файла (столбец **Logical Name**), начальный размер файла (**Initial Size (MB)**) и характеристики увеличения размера (**Autogrowth**). А вот изменить имя физического файла или путь к физическому файлу в диалоговом режиме Management Studio нельзя, поскольку такие изменения требуют перевода базы данных в **OFFLINE**, а в этом состоянии базы невозможен просмотр ее свойств.

Сейчас удалить ни один из существующих файлов базы **BestDatabase** мы не можем, потому что она содержит минимальное количество требуемых файлов — один файл данных и один файл журнала транзакций. По этой причине кнопка **Remove** (удалить) является неактивной.

Для изменения логического имени файла (данных или журнала транзакций) нужно щелкнуть мышью по имени соответствующего файла. Поле станет доступным для изменения. После внесения изменений нужно нажать клавишу **<Enter>**. Надо сказать, что система не проверяет вводимые символы, вы можете помещать в имя пробелы, вообще любые символы, что одобрить, как вы понимаете, нельзя.

Для изменения начального размера файла нужно мышью выделить соответствующий элемент (заголовок столбца **Initial Size (MB)**). В правой части строки появится управляющий элемент, позволяющий изменять размер на 1 Мбайт в сторону увеличения или уменьшения (рис. 3.25). Размер также можно менять, вводя с клавиатуры в этом поле нужное значение.

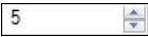


Рис. 3.25. Изменение начального размера файла

После внесения изменения следует перевести фокус ввода на любой другой элемент.

Для изменения характеристики увеличения размера (столбец **Autogrowth**) щелкните мышью по кнопке с многоточием в правой части этого поля у нужного файла. Появится диалоговое окно, позволяющее изменить характеристики увеличения размера файла. На рис. 3.26 показано такое диалоговое окно для файла журнала транзакций базы данных **BestDatabase**.

Здесь установлен флажок допустимости автоматического увеличения размера (**Enable Autogrowth**). Если увеличение размера возможно, то в группе переключателей **File Growth** (увеличение размера файла) можно указать, в каких единицах размер файла должен увеличиваться: в процентах (**In Percent**) или в мегабайтах (**In Megabytes**).

Можно задать максимальный размер файла в группе переключателей **Maximum File Size**: максимальный (ограниченный) размер файла в мегабайтах (**Restricted File Growth (MB)**) или неограниченный размер файла (**Unrestricted File Growth**).

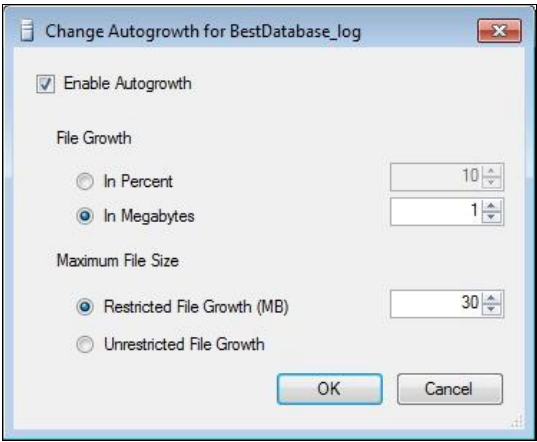


Рис. 3.26. Изменение характеристик увеличения размера файла

Для добавления нового файла (данных или журнала транзакций) нужно на вкладке **Files** щелкнуть по кнопке **Add** (добавить). В список файлов будет добавлена новая строка, где отдельные столбцы будут содержать пустые значения, а для других установлены значения по умолчанию (рис. 3.27).

Logical Name	File Type	Filegroup	Initi...	Autogrowth / Maxsize	Path	File Name
BestDatabase_dat	Rows Data	PRIMARY	5	By 1 MB, Unlimited	...	D:\BestDatabase Winner.mdf
BestDatabase_log	Log	Not Applicable	1	By 10 percent, Limited to 2097152 MB	...	D:\BestDatabase Winner.ldf
	Rows Data	PRIMARY	5	By 1 MB, Unlimited	...	C:\Program File...

Рис. 3.27. Добавление нового файла в базу данных BestDatabase

Вначале добавим файл данных. Зададим ему логическое имя (**Logical Name**). Щелкните мышью по пустому значению этого поля и введите логическое имя файла, например, `B2_dat`. Пусть это будет второй файл данных в нашей базе данных `BestDatabase`. По этой причине поле **File Type** (тип файла) мы не меняем, а оставляем значение, установленное по умолчанию, — `Rows Data` (строки данных, т. е. файл данных). Значение поля **Filegroup** (файловая группа), заданное как `PRIMARY`, не изменяем. Начальный размер файла, установленный в 5 Мбайт по умолчанию (поле **Initial Size (MB)**), также оставим без изменения. Не станем изменять и величину автоматического увеличения размера файла (**Autogrowth**). А вот новый путь к файлу, отличный от пути по умолчанию, зададим. Это можно сделать двумя способами. Первый — для любителей все вводить руками. Нужно щелкнуть мышью по полю **Path** и с клавиатуры набрать полный путь к файлу, не забыв указать и имя внешнего устройства. Напомню, что все каталоги в этом пути уже должны существовать на выбранном вами носителе.

В случае использования диалоговых средств нужно просто щелкнуть мышью по кнопке с многоточием справа от заданного пути по умолчанию в поле **Path**. Появится окно выбора пути для размещения этого файла (рис. 3.28). Выберите диск `D:` и уже существующий наш каталог `BestDatabase` и щелкните по кнопке **OK**.

В поле имени файла (**File Name**) введем имя создаваемого файла `Winner2.ndf`.

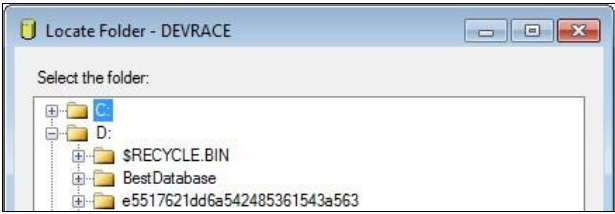


Рис. 3.28. Выбор папки для размещения нового файла базы данных

Второй файл данных для нашей базы данных BestDatabase мы создали. Теперь создадим второй файл журнала транзакций.

Щелкните в окне по кнопке **Add**. В списке файлов появится новая строка. Задайте логическое имя файла B2_log. Щелкните по полю типа файла (**File Type**) и из раскрывающегося списка выберите значение Log.

Выберите каталог для размещения файла D:\BestDatabase. Имя файла введите Winner2.ldf.

Теперь новый список файлов этой базы данных будет выглядеть следующим образом (рис. 3.29).

Logical Name	File Type	Filegroup	Initi...	Autogrowth / Maxsize	Path	File Name
BestDatabase_dat	Rows Data	PRIMARY	5	By 1 MB, Unlimited	D:\BestDatabase	Winner.mdf
BestDatabase_log	Log	Not Applicable	1	By 10 percent, Limited to 2097152 MB	D:\BestDatabase	Winner.ldf
B2_dat	Rows Data	PRIMARY	5	By 1 MB, Unlimited	D:\BestDatabase	Winner2.ndf
B2_log	Log	Not Applicable	1	By 10 percent, Unlimited	D:\BestDatabase	Winner2.ldf

Рис. 3.29. Добавленные два файла в базу данных BestDatabase

Чтобы созданные в предыдущих шагах добавления были фактически выполнены для текущей базы данных, щелкните по кнопке **OK**. Если при создании любого из добавляемых файлов вы допустили ошибки, то сообщения о них появятся только сейчас. Например, если вы задали имя для второго файла журнала транзакций такое же, как и для первого файла, то после щелчка по кнопке **OK** вы получите такое сообщение, как на рис. 3.30.

Здесь говорится о том, что файл 'D:\BestDatabase\Winner2.ldf' не может быть перезаписан, поскольку он используется базой данных BestDatabase.

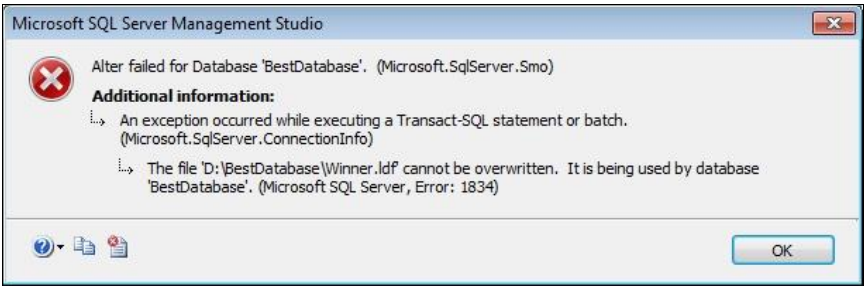


Рис. 3.30. Сообщение о дублировании имен файлов

Для удаления любого файла базы данных (кроме первичного) нужно на вкладке **Files** щелкнуть мышью по кнопке **Remove**, расположенной в нижней части окна. Текущий (выделенный в окне) файл будет удален.

ВНИМАНИЕ!
При щелчке по кнопке удаления система не запрашивает подтверждения необходимости удаления текущего файла. Однако если вам на самом деле не нужно удалять этот файл, вы просто в окне просмотра свойств базы данных можете щелкнуть по кнопке **Cancel** (отмена всех выполненных изменений базы данных).

Если вы выделяете на форме файл, который не может быть удален, то кнопка **Remove** будет в недоступном состоянии.

3.5.2.3. Изменение файловых групп базы данных

Для изменения файловых групп базы BestDatabase щелкните мышью по строке **Filegroups** в панели **Select a Page**. Появится список файловых групп (рис. 3.31).

Name	Files	Read-Only	Default
PRIMARY	3		<input checked="" type="checkbox"/>

Рис. 3.31. Список файловых групп базы данных BestDatabase

Пока база данных содержит только одну первичную (PRIMARY) файловую группу. Чтобы добавить вторую файловую группу, нужно щелкнуть мышью по кнопке **Add** в нижней части этого окна. В списке появится новая строка. В поле имени (**Name**) введите имя файловой группы, например, SECOND (рис. 3.32).

Name	Files	Read-Only	Default
PRIMARY	3		<input checked="" type="checkbox"/>
SECOND	0	<input type="checkbox"/>	<input type="checkbox"/>

Рис. 3.32. Измененный список файловых групп базы данных BestDatabase

Видно, что количество файлов, принадлежащих этой группе (поле **Files**), равно нулю. Чтобы переместить только что созданный новый файл данных B2_dat в эту файловую группу, щелкните мышью по строке **Files** в панели **Select a Page**.

В столбце **Filegroup** для файла B2_dat из раскрывающегося списка выберите имя файловой группы SECOND.

Этот файл станет принадлежать файловой группе SECOND.

ВНИМАНИЕ!

Поместить во вновь созданную файловую группу можно только созданные в этой же сессии файлы, только те файлы, которые были созданы до щелчка по кнопке **ОК**.

Давайте для порядка создадим еще один файл данных и одновременно с этим для него новую файловую группу.

На вкладке просмотра файлов щелкните по кнопке **Add**. Создайте новый файл данных с любыми характеристиками (мы его потом удалим вместе с новой файловой группой). При задании файловой группы (столбец **Filegroup**) из раскрывающегося списка выберите последнюю строку <new filegroup>. Это означает, что для файла вы собираетесь создать новую файловую группу. Появится диалоговое окно задания характеристик файловой группы (рис. 3.33).

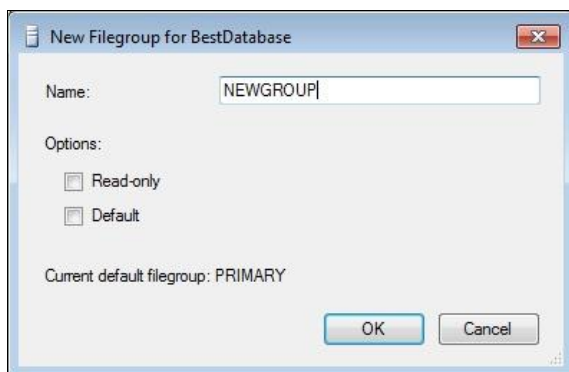


Рис. 3.33. Создание новой файловой группы

В поле **Name** введите имя группы, например NEWGROUP, и щелкните по кнопке **ОК**, чтобы создать эту группу.

В этом диалоговом окне существует еще два флажка — **Read-only** (только для чтения) и **Default** (файловая группа по умолчанию). Ни тот, ни другой мы устанавливать не будем. Во-первых, мы предполагаем, что для файлов, включаемых в эту группу, будут допустимы не только операции чтения, но и записи, во-вторых, мы не собираемся менять файловую группу по умолчанию; такой у нас является группа PRIMARY.

Перейдите во вкладку **Filegroups**. Вы увидите в списке файловых групп и только что созданную группу.

Чтобы удалить любую файловую группу, кроме первичной, нужно выделить ее в списке мышью и щелкнуть по кнопке **Remove**. Будет удалена файловая группа, а все файлы, принадлежащие этой группе, будут автоматически перемещены в первичную файловую группу.

Удалите только что созданную группу. Вы увидите, что принадлежащий ей файл будет перемещен в группу `PRIMARY`.

3.5.2.4. Изменение других характеристик базы данных

Общие характеристики базы данных можно изменять, если в окне просмотра свойств этой базы данных в панели **Select a Page** выбрать вкладку **Options** (рис. 3.34).

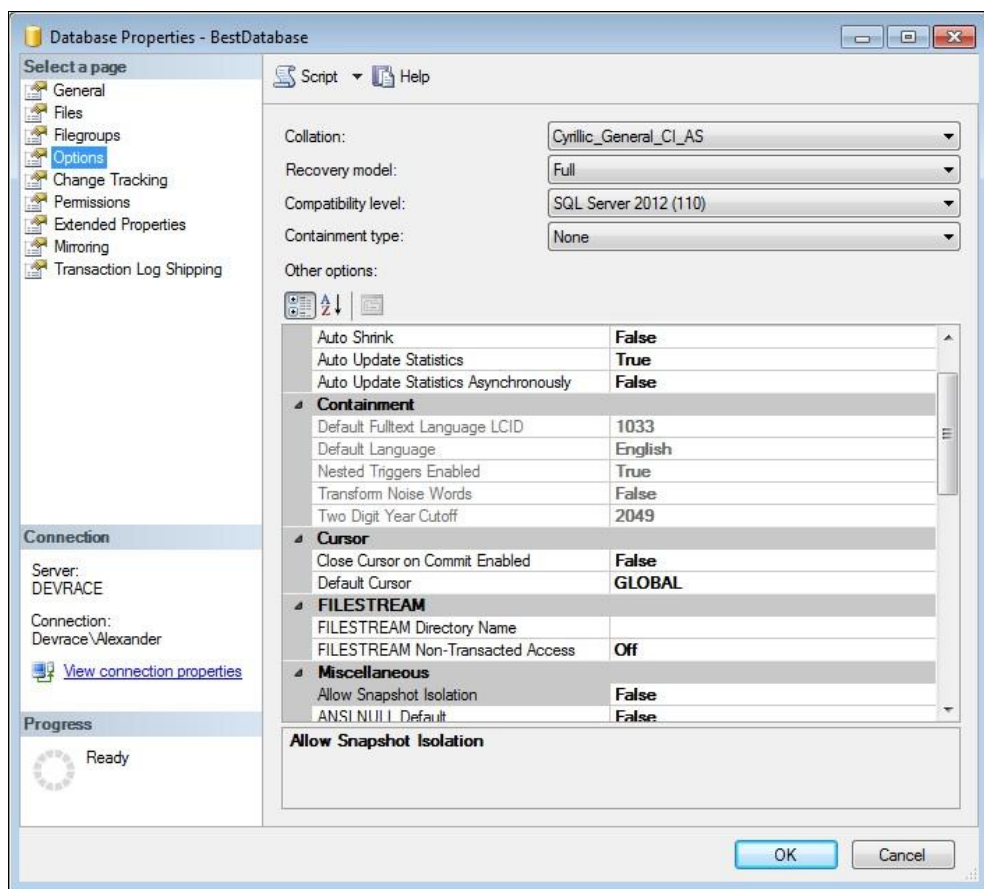


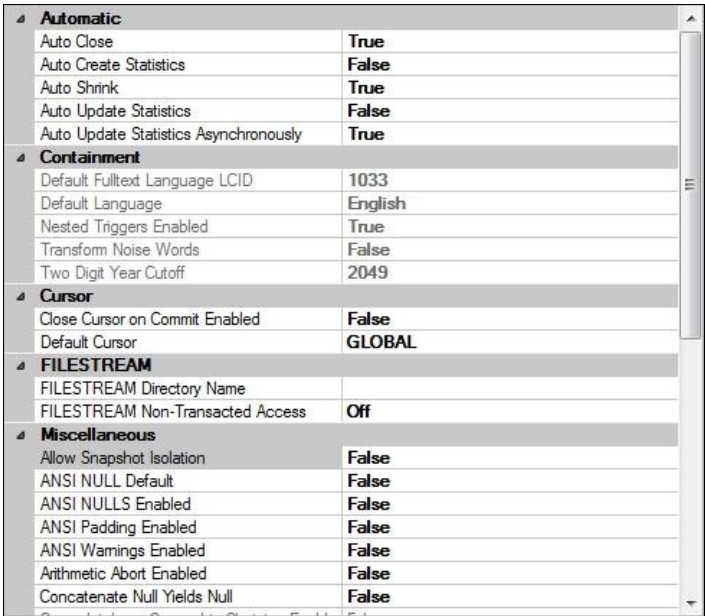
Рис. 3.34. Вкладка свойств базы данных

Во-первых, здесь можно изменить набор символов для базы данных, выбрав в раскрывающемся списке **Collation** нужный набор. Заменяем установленный нами по умолчанию при создании базы данных порядок сортировки `Cyrillic_General_CI_AS` на порядок сортировки, позволяющий вводить символы, присутствующие, скажем, во французском языке. Помимо обычных латинских букв в этом языке также существуют буквы `ç`, `é`, `è` и ряд других. Щелкните по текущему элементу списка **Collation**. Появится раскрывающийся список доступных порядков сортировки. Выберите из списка `French_CI_AI`.

Можно изменить модель восстановления. Для этого нужно в раскрывающемся списке **Recovery model** выбрать требуемую модель: **Full**, **Bulk-logged**, **Simple**.

Существует понятие уровня совместимости с предыдущими версиями: **Compatibility level**. Для вновь создаваемых систем обработки данных следует оставить для уровня совместимости значение по умолчанию.

В основном списке режимов (рис. 3.35) присутствует множество характеристик базы данных в целом. Те характеристики, которые можно изменять на этой форме, представлены шрифтом обычного цвета. Если характеристику изменять нельзя, то соответствующая строка имеет серый шрифт.

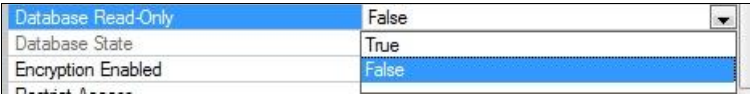


Automatic	
Auto Close	True
Auto Create Statistics	False
Auto Shrink	True
Auto Update Statistics	False
Auto Update Statistics Asynchronously	True
Containment	
Default Fulltext Language LCID	1033
Default Language	English
Nested Triggers Enabled	True
Transform Noise Words	False
Two Digit Year Cutoff	2049
Cursor	
Close Cursor on Commit Enabled	False
Default Cursor	GLOBAL
FILESTREAM	
FILESTREAM Directory Name	
FILESTREAM Non-Transacted Access	Off
Miscellaneous	
Allow Snapshot Isolation	False
ANSI NULL Default	False
ANSI NULLS Enabled	False
ANSI Padding Enabled	False
ANSI Warnings Enabled	False
Arithmetic Abort Enabled	False
Concatenate Null Yields Null	False

Рис. 3.35. Общие свойства базы данных

Чтобы изменить значение характеристики, нужно щелкнуть мышью по этой строке; в правой части значения характеристики появится стрелка, позволяющая вызвать соответствующий раскрывающийся список. Чаще всего в списке присутствует только два значения — истина и ложь (**True** и **False**). На рис. 3.36 показан пример появления раскрывающегося списка с двумя значениями **True** и **False** для свойства **Database Read-Only**.

Если этому свойству задать значение **True**, то база данных становится базой только для чтения (база данных **READ_ONLY**). Если значение установлено в **False** (значение



Database Read-Only	False
Database State	True
Encryption Enabled	False

Рис. 3.36. Выбор значения для свойства базы данных **Database Read-Only**

при создании базы данных по умолчанию), то для базы допустимы как операции чтения, так и записи, т. е. добавление, изменение и удаление данных (база данных `READ_WRITE`).

ВНИМАНИЕ!

Еще раз напомним. Чтобы проделанная вами работа по изменению любых характеристик базы, ее файлов и файловых групп действительно была зафиксирована в базе данных, необходимо в окне просмотра свойств базы в завершение всех действий щелкнуть по кнопке **ОК**. Только тогда изменения будут применены к базе данных. Одна моя знакомая (вы не поверите — она натуральная блондинка) жаловалась мне, что не может в Management Studio изменить нужные ей характеристики. Оказалось, что после выполнения изменений она просто закрывала окно, щелкая по кнопке закрытия в правом верхнем углу этого окна.

В этой грустной истории есть один полезный для нас с вами смысл. Вы можете вносить любые изменения в параметры базы данных. Потом, если вдруг замечаете, что сделали совсем не то, что было нужно, вы спокойно отмените все эти безобразия, щелкнув по кнопке **Cancel** или просто закрыв окно.

В Management Studio можно очень просто переводить базу данных в неактивное (`OFFLINE`) и активное (`ONLINE`) состояние.

Если вам понадобится скопировать базу данных на другой носитель, то вы не сможете этого сделать, если при запущенном на выполнении сервере базы данных Database Engine база находится в состоянии `ONLINE`. Для этих целей базу нужно перевести в состояние `OFFLINE`.

Чтобы перевести базу данных в неактивное состояние, нужно в панели **Object Explorer** щелкнуть правой кнопкой мыши по имени базы, в контекстном меню выбрать элемент **Tasks** (задачи) и в открывшемся подменю **Take Offline**. Для перевода в активное состояние в этом подменю нужно выбрать элемент **Bring Online**.

3.5.2.5. Отображение отчета использования дискового пространства базы данных

Здесь хочу сказать два слова о тех "красивостях", которые можно получать, работая с SQL Server. В Management Studio можно создавать различные отчеты, отображающие состояние базы данных и ее объектов. Давайте сейчас создадим отчет, отображающий использование дискового пространства файлами базы данных BestDatabase. На панели **Object Explorer** щелкните правой кнопкой мыши по имени базы данных, в контекстном меню выберите элемент **Reports**, затем элементы **Standard Reports** и **Disk Usage**. В результате будет создан наглядный отчет, отображающий использование базой данных внешнего пространства (рис. 3.37).

В процессе заполнения базы данных необходимыми данными можно периодически создавать такой отчет, который будет отображать текущее положение с использованием дискового пространства.