|  |
| --- |
| EPAM Systems |
| **Microsoft SQL Server 2005**  **SFT.CM.04 MS SQL Server** |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **REVISION HISTORY** | | | | | |
| **Rev.** | **Description of Change** | **Author** | **Date** | **Approved** | |
| **Name** | **Effective Date** |
| 1.0 | Начальная версия | Степанов Андрей | 2.12.2008 |  |  |
| 1.1 | Minor corrections | Горбачев Клим | 6.08.2014 |  |  |
| 1.2 | Errors correction | Оксана Небогаткина | 19.06.2017 |  |  |

|  |  |
| --- | --- |
| **RELATED ARTIFACTS** | |
| **No.** | **Artifact Name** |
|  |  |
|  |  |

[**1.**](#_1xrdshw) **ПРЕДМЕТ КУРСА 4**

[**2.**](#_3dy6vkm) **ПОЯСНЕНИЯ К ПРОГРАММЕ КУРСА 4**

[**3.**](#_1t3h5sf) **ТЕОРЕТИЧЕСКИЙ РАЗДЕЛ 4**

[3.1.](#_4d34og8) Реляционная модель данных 4

[*3.1.1.*](#_2s8eyo1) *Определение реляционной модели данных 4*

[*3.1.2.*](#_3rdcrjn) *Связывание таблиц 7*

[3.2.](#_4hr1b5p) Основы проектирования баз данных 12

[*3.2.1.*](#_2wwbldi) *Целостность данных 12*

[*3.2.2.*](#_1c1lvlb) *Целостность сущностей 12*

[*3.2.3.*](#_3w19e94) *Целостность домена 12*

[*3.2.4.*](#_2b6jogx) *Ссылочная целостность 12*

[*3.2.5.*](#_qbtyoq) *Определенная пользователем целостность 13*

[*3.2.6.*](#_3abhhcj) *Целостность транзакций 13*

[*3.2.7.*](#_1pgrrkc) *Ошибки транзакций 14*

[*3.2.8.*](#_49gfa85) *Пустые значения 16*

[*3.2.9.*](#_2olpkfy) *Индексация 16*

[3.3.](#_13qzunr) Введение в Microsoft SQL Server 2005 16

[*3.3.1.*](#_3nqndbk) *Службы SQL Server 17*

[*3.3.2.*](#_22vxnjd) *Реляционное ядро 17*

[*3.3.3.*](#_i17xr6) *Transact-SQL 17*

[*3.3.4.*](#_320vgez) *Visual Studio и CLR 18*

[*3.3.5.*](#_1h65qms) *Служба репликаций 18*

[*3.3.6.*](#_415t9al) *Брокер служб 18*

[*3.3.7.*](#_2gb3jie) *Полнотекстовый поиск 19*

[*3.3.8.*](#_vgdtq7) *Служба уведомлений 19*

[*3.3.9.*](#_3fg1ce0) *SQL Server Agent 19*

[*3.3.10.*](#_1ulbmlt) *Координатор распределенных транзакций 19*

[*3.3.11.*](#_4ekz59m) *SQL Mail 19*

[*3.3.12.*](#_2tq9fhf) *Службы бизнес-аналитики 20*

[*3.3.13.*](#_18vjpp8) *Служба интеграции 20*

[*3.3.14.*](#_3sv78d1) *Служба отчетности 20*

[*3.3.15.*](#_280hiku) *Служба анализа 20*

[*3.3.16.*](#_n5rssn) *Различные редакции SQL Server 2005 20*

[*3.3.17.*](#_375fbgg) *Утилиты и компоненты SQL Server 22*

[*3.3.18.*](#_1maplo9) *Обзор метаданных 22*

[3.4.](#_46ad4c2) Программирование на языке Transact-SQL 23

[*3.4.1.*](#_2lfnejv) *Основы Transact-SQL 24*

[*3.4.2.*](#_10kxoro) *Реализация физической схемы базы данных 25*

[*3.4.3.*](#_3kkl7fh) *Создание таблиц 30*

[*3.4.4.*](#_1zpvhna) *Создание первичных ключей 31*

[*3.4.5.*](#_4jpj0b3) *Создание внешних ключей 34*

[*3.4.6.*](#_2yutaiw) *Создание пользовательских столбцов данных 37*

[*3.4.7.*](#_1e03kqp) *Типы данных столбцов 37*

[*3.4.8.*](#_3xzr3ei) *Ограничения и значения столбцов по умолчанию 40*

[*3.4.9.*](#_2d51dmb) *Манипулирование данными с помощью инструкции SELECT 42*

[*3.4.10.*](#_sabnu4) *Использование объединений 49*

[*3.4.11.*](#_3c9z6hx) *Модификация данных 54*

[3.5.](#_1rf9gpq) Представления 56

[*3.5.1.*](#_4bewzdj) *Работа с представлениями 56*

[3.6.](#_2qk79lc) Хранимые процедуры 57

[*3.6.1.*](#_15phjt5) *Инструкции CREATE, ALTER и DROP 57*

[*3.6.2.*](#_3pp52gy) *Возвращение набора записей 57*

[*3.6.3.*](#_24ufcor) *Компиляция хранимых процедур 58*

[*3.6.4.*](#_jzpmwk) *Передача данных в хранимые процедуры 58*

[*3.6.5.*](#_33zd5kd) *Получение данных из хранимой процедуры 58*

[3.7.](#_1j4nfs6) Триггеры 58

[*3.7.1.*](#_434ayfz) *Порядок выполнения транзакций 59*

[*3.7.2.*](#_2i9l8ns) *Создание триггеров 60*

[*3.7.3.*](#_xevivl) *Триггеры AFTER 60*

[*3.7.4.*](#_3hej1je) *Триггеры INSTEAD OF 60*

[*3.7.5.*](#_1wjtbr7) *Работа с транзакциями 60*

[3.8.](#_4gjguf0) Курсор 61

[*3.8.1.*](#_2vor4mt) *Пять этапов жизни курсора 61*

[*3.8.2.*](#_1au1eum) *Управление курсором 62*

[3.9.](#_3utoxif) Выполнение массовых операций 62

[*3.9.1.*](#_29yz7q8) *Команда bulk insert 62*

[*3.9.2.*](#_p49hy1) *Утилита BCP 63*

[3.10.](#_393x0lu) Измерение производительности 63

[*3.10.1.*](#_1o97atn) *Использование SQL Server Profiler 63*

[**4.**](#_ymfzma) **ИСТОЧНИКИ 65**

[**5.**](#_3im3ia3) **СПИСОК РЕКОМЕНДОВАННОЙ ЛИТЕРАТУРЫ ДЛЯ ДОПОЛНИТЕЛЬНОГО ИЗУЧЕНИЯ 65**

# **Предмет курса**

Целями данного курса является:

* Общее знакомство с компонентами и архитектурой SQL Server 2005
* Ознакомиться с набором инструментальных средств SQL Server 2005, и на практике ознакомиться с основными из них - SQL Server Management Studio и SQL Server Profiler
* Изучить основные возможности SQL Server Management Studio (работа с различными панелями, выполнение операторов Transact SQL, просмотр результирующих наборов, планов исполнения, трассировочные сведения, статистку и сообщения, использование отладчика)
* Изучить и ознакомиться с основными операторами DML языка Transact SQL

# **Пояснения к программе курса**

По окончании изучения теоретического материала программой курса предусматривается выполнение практического задания, основанного на изученных темах.

Умения, которые должны быть продемонстрированы при выполнении практической части:

* Практические навыки работы с SQL Server Management Studio.
* Разработка разнообразных SQL запросов с использованием операторов языка Transact SQL.

# **Теоретический раздел**

## **Реляционная модель данных**

### *Определение реляционной модели данных*

**Реляционная модель данных** (РМД) некоторой предметной области представляет собой набор отношений, изменяющихся во времени. При создании информационной системы совокупность отношений позволяет хранить данные об объектах предметной области и моделировать связи между ними. Элементы реляционной модели данных и формы их представления приведены в Табл. 1.

**Табл. 1. Элементы реляционной модели**

|  |  |
| --- | --- |
| **Элемент реляционной модели** | **Форма представления** |
| Отношение | Таблица |
| Схема отношения | Строка заголовков столбцов таблицы (заголовок таблицы) |
| Кортеж | Строка таблицы |
| Сущность | Описание свойств объекта |
| Атрибут | Заголовок столбца таблицы |
| Домен | Множество допустимых значений атрибута |
| Значения атрибута | Значения поля в записи |
| Первичный ключ | Один или несколько атрибутов |
| Тип данных | Тип значений элементов таблицы |

**Отношение** является важнейшим понятием и представляет собой двумерную таблицу, содержащую некоторые данные.

**Сущность** есть объект любой природы, данные о котором хранятся в базе данных. Данные о сущности хранятся в отношении.

**Атрибуты** представляют собой свойства, характеризующие сущность. В структуре таблицы каждый атрибут именуется и ему соответствует заголовок столбца таблицы.

На Рис. 1 приведен пример представления отношения СОТРУДНИК.



|  |  |  |  |
| --- | --- | --- | --- |
| **ФИО** | **Отдел** | **Должность** | **Д\_рождения** |
| Иванов И.И. | 002 | Начальник | 27.08.51 |
| Петров П.П. | 001 | Заместитель | 15.04.55 |
| Сидоров И.П. | 002 | Инженер | 13.01.70 |

В общем случае порядок кортежей в отношении, как и в любом множестве, не определен. Однако в реляционных СУБД для удобства кортежи все же упорядочивают. Чаще всего для этого выбирают некоторый атрибут, по которым система автоматически сортирует кортежи по возрастанию или убыванию. Если пользователь не назначает атрибута упорядочения, система автоматически присваивает номер кортежам в порядке их ввода.

**Домен** представляет собой множество всех возможных значений определенного атрибута отношения. Отношение СОТРУДНИК включает 4 домена. *Домен 1* содержит фамилии всех сотрудников, *домен 2* – номера всех отделов фирмы, *домен 3* – название всех должностей, *домен 4* – даты рождения всех сотрудников. Каждый домен образует значения одного типа данных, например, числовые или символьные.

Отношение СОТРУДНИК содержит 3 кортежа. Кортеж рассматриваемого отношения состоит из 4 элементов, каждый из которых выбирается из соответствующего домена. Каждому кортежу соответствует строка таблицы (Рис. 1).

**Схема отношения (заголовок отношения)** представляет собой список имен атрибутов. Например, для приведенного примера схема отношения имеет вид СОТРУДНИК(ФИО, Отдел, Должность, Д\_Рождения). Множество собственно кортежей отношения часто называют **содержимым (телом) отношения**.

**Первичным ключом (ключом отношения, ключевым атрибутом)** называется атрибут отношения, однозначно идентифицирующий каждый из его кортежей. Например, в отношении СОТРУДНИК(ФИО, Отдел, Должность, Д\_Рождения) ключевым является атрибут «ФИО». Ключ может быть **составным (сложным)**, то есть состоять из нескольких атрибутов.

Каждое отношение обязательно имеет комбинацию атрибутов, которая может служить ключом. Ее существование гарантируется тем, что отношение – это множество, которое не содержит одинаковых элементов – кортежей. То есть в отношении нет повторяющихся кортежей, а это значит, что, по крайней мере, вся совокупность атрибутов обладает свойством однозначной идентификации кортежей отношения.

Возможно случаи, когда отношение имеет несколько комбинаций атрибутов, каждая из которых однозначно определяет все кортежи отношения. Все эти комбинации атрибутов являются **возможными ключами** отношения. Любой из возможных ключей может быть выбран как *первичный*.

Если выбранный первичный ключ состоит из минимально необходимого набора атрибутов, говорят, что он является **не избыточным**.

Ключи обычно используют для достижения следующих целей:

1. Исключения дублирования значений в ключевых атрибутах (остальные атрибуты в расчет не принимаются);
2. Упорядочения кортежей. Возможно упорядочение по возрастанию или убыванию значений всех ключевых атрибутов, а также смешанное упорядочение (по одним – возрастание, а по другим – убывание);
3. Ускорения работы с кортежами отношения;
4. Организация связывания таблиц;

Пусть в отношении R1 имеется *не ключевой* атрибут A, значения которого являются значениями *ключевого* атрибута B другого отношения R2. Тогда говорят, что атрибут A отношения R1 есть **внешний ключ**.

С помощью внешних ключей устанавливаются связи между отношениями. Например, имеются два отношения СТУДЕНТ(ФИО, Группа, Специальность) и ПРЕДМЕТ(Назв.Пр, Часы), которые связаны отношением СТУДЕНТ\_ПРЕДМЕТ(ФИО, Назв.Пр, Оценка) (Рис. 2). В связующем отношении атрибуты ФИО и Назв.Пр образуют составной ключ. Эти атрибуты представляют собой внешние ключи, являющиеся первичными ключами других отношений.



|  |  |  |
| --- | --- | --- |
| ФИО | Группа | Специальность |
| Назв.Пр | Часы |
| ФИО | Назв.Пр | Оценка |

Реляционная модель накладывает на внешние ключи ограничения для обеспечения целостности данных, называемое ссылочной целостностью. Это означает, что каждому значению внешнего ключа должны соответствовать строки в связанных отношениях.

Поскольку не всякой таблице можно поставить в соответствие отношение, приведем условия, выполнения которых позволяет таблицу считать отношением.

1. Все строки таблицы должны быть уникальны, то есть не может быть строк с одинаковыми первичными ключами.
2. Имена столбцов таблицы должны быть различны, а значения их простыми, то есть недопустима группа значений в одном столбце одной строки.
3. Все строки одной таблицы должны иметь одну структуру, соответствующую именам и типам столбцов.
4. Порядок размещения строк в таблице может быть произвольным.

Если задаваемое таблицей отношение имеет ключ, то считается, что таблица тоже имеет ключ, и ее называют **ключевой** или **таблицей с ключевыми полями**.

### *Связывание таблиц*

При проектировании реальных БД информацию обычно размещают в нескольких таблицах. Таблицы при этом связаны семантикой информации. В реляционных СУБД для указания связей таблиц производят операцию их связывания.

Укажем выигрыш, обеспечиваемый в результате связывания таблиц. Многие СУБД при связывании таблиц автоматически выполняют контроль целостности вводимых в базу данных в соответствии с установленными связями. В конечном итоге это повышает достоверность хранимой в БД информации.

Кроме того, установление связи между таблицами облегчает доступ к данным. Связывание таблиц при выполнении таких операций, как поиск, просмотр, редактирование, выборка и подготовка отчетов, обычно обеспечивает возможность обращения к произвольным полям связанных записей. Это уменьшает количество явных обращений к таблицам данных и число манипуляций в каждой из них.

#### Основные виды связи таблиц

Между таблицами могут устанавливаться бинарные (между двумя таблицами), тернарные (между тремя таблицами) и, в общем случае, n-арные связи. Рассмотрим наиболее часто встречающиеся *бинарные* связи.

При связывании двух таблиц выделяют основную и дополнительную (подчиненную) таблицы. Логическое связывание таблиц производится с помощью **ключа связи**.

Ключ связи, по аналогии с обычным ключом таблицы, состоит из одного или нескольких полей, которые в данном случае называют **полями связи** (ПС).

Суть связывания состоит в установлении соответствия полей связи основной и дополнительной таблиц. Поля связи основной таблицы могут быть обычными и ключевыми. В качестве полей связи подчиненной таблицы чаще всего используют ключевые поля.

В зависимости от того, как определены поля связи основной и дополнительной таблиц (как соотносятся ключевые поля с полями связи), между двумя таблицами в общем случае могут устанавливаться следующие четыре основных вида связи (Табл. 2):

* один – один (1:1);
* один – много (1:М);
* много – один (М:1);
* много – много (М:М или М:N).

**Табл. 2. Характеристика видов связей таблиц**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Характеристика полей связи по видам** | **1:1** | **1:M** | **M:1** | **M:M** |
| **Поля связи основной таблицы** | являются ключом | являются ключом | не являются ключом | не являются ключом |
| **Поля связи дополнительной таблицы** | являются ключом | не являются ключом | являются ключом | не являются ключом |

Дадим характеристику названным видам связи между двумя таблицами и приведем примеры их использования.

##### Связь вида 1:1

Связь вида 1:1 образуется в случае, когда все поля связи основной и дополнительной таблиц являются ключевыми. Поскольку значения в ключевых полях обеих таблиц не повторяются, обеспечивается взаимно-однозначное соответствие записей из этих таблиц. Сами таблицы, по сути, здесь становятся равноправными.

*Пример.*

Пусть имеются сведения о выполняемых в некоторой организации научно-исследовательских работах. Эти данные включают в себя следующую информацию по каждой из работ: тему (девиз и полное наименование работ), шифр (код), даты начала и завершения работы, количества этапов, головного исполнителя и другую дополнительную информацию. Все работы имеют гриф «Для служебного пользования» или «секретно».

В такой ситуации всю информацию целесообразно хранить в двух таблицах: в одной из них – всю секретную информацию (например, шифр, полное наименование работы и головной исполнитель), а в другой – всю оставшуюся несекретную информацию. Обе таблицы можно связать по шифру работы. Первую из таблиц целесообразно защитить от несанкционированного доступа.

##### Связь вида 1:М

Связь 1:М имеет место в случае, когда одной записи основной таблицы соответствует несколько записей вспомогательной таблицы.

*Пример.*

Пусть имеются две связанные таблицы О2 и Д2. В Табл. 3. O2 содержится информация о видах мультимедиа-устройств ПЭВМ, а в Табл. 4. Д2 – сведения о фирмах-производителях этих устройств, а также о наличии на складе хотя бы одного устройства.

**Табл. 3. O2**

|  |  |
| --- | --- |
| \* + |  |
| Код | Вид устройства |
| а | CD-ROM |
| б | CD-Recorder |
| в | Sound Blaster |

**Табл. 4. Д2**

|  |  |  |
| --- | --- | --- |
| Код | Фирма-производитель | Наличие |
| а | Acer | да |
| а | Mitsumi | нет |
| а | NEC | да |
| а | Panasonic | да |
| а | Sony | да |
| б | Philips | нет |
| б | Sony | нет |
| б | Yamaha | да |
| в | Creative Labs | да |

Таблица Д2 имеет два ключевых поля, так как одна и та же фирма может производить устройства различных видов. В примере фирма Sony производит устройства считывания и перезаписи с компакт-дисков.

Сопоставление записей обеих таблиц по полю «Код» порождает псевдозаписи вида: (а, CD-ROM, Acer, да), (а, CD-ROM, Mitsumi, нет), (а, CD-ROM, NEC, да), (а, CD-ROM, Panasonic, да), (а, CD-ROM, Sony, да), (б, CD-Recorder, Philips, нет), (б, CD-ROM, Sony, да) и т.д.

Если свести псевдозаписи в новую таблицу, то получим полную информацию обо всех видах мультимедиа-устройств ПЭВМ, фирмах, их производящих, а также сведения о наличии конкретных видов устройств на складе.

##### Связь вида М:1

Связь М:1 имеет место в случае, когда одной или нескольким записям основной таблицы ставится в соответствии одна запись дополнительной таблицы.

*Пример.*

Рассмотрим связь таблиц О3 и Д3. В основной таблице О3 содержится информация о названиях деталей (Поле11), видах материалов, из которого детали можно изготовить (Поле12), и марках материала (Поле13).

В дополнительной таблице Д3 содержатся сведения о названиях деталей (Поле21), планируемых сроках изготовления (Поле22) и стоимости заказов (Поле23).

**Табл. 5. О3**

|  |  |  |
| --- | --- | --- |
| + |  |  |
| Поле11 | Поле12 | Поле13 |
| деталь1 | чугун | марка1 |
| деталь1 | чугун | марка2 |
| деталь2 | сталь | марка1 |
| деталь2 | сталь | марка2 |
| деталь2 | сталь | марка3 |
| деталь3 | алюминий | - |
| деталь4 | чугун | марка2 |

**Табл. 6. Д3**

|  |  |  |
| --- | --- | --- |
| \* + |  |  |
| Поле21 | Поле22 | Поле23 |
| деталь1 | 4.03.98 | 90 |
| деталь2 | 3.01.98 | 35 |
| деталь3 | 17.02.98 | 90 |
| деталь4 | 6.05.98 | 240 |

Связывание этих таблиц обеспечивает такое установление соответствия между записями, которое эквивалентно образованию следующих псевдозаписей: (деталь1, чугун, марка1, 4.03.98, 90), (деталь1, чугун, марка2, 4.03.98, 90), (деталь2, сталь, марка1, 3.01.98, 35), (деталь2, сталь, марка2, 3.01.98, 35), (деталь2, сталь, марка3, 3.01.98, 35), (деталь3, алюминий, - , 17.02.98, 90), (деталь4, чугун, марка2, 6.05.98, 240).

Полученная псевдотаблица может быть полезна при планировании или принятии управленческих решений, когда необходимо иметь все возможные варианты исполнения заказов по каждому изделию. Отметим, что таблица О3 не имеет ключей и в ней возможно повторение записей. Если таблицу Д3 сделать основной, а таблицу О3 – дополнительной, получим связь вида 1:М. Поступив аналогично с таблицами О2 и Д2, можно получить связь вида М:1. Отсюда следует, что вид связи (1:М или М:1) зависит от того, какая таблица является главной, а какая дополнительной.

##### Связь вида M:M

Самый общий вид связи M:M возникает в случаях, когда нескольким записям основной таблицы соответствует несколько записей дополнительной таблиц.

*Пример.*

Пусть в основной таблице О4 содержится информация о том, на каких станках могут работать рабочие некоторой бригады. Таблица Д4 содержит сведения о том, кто из бригады ремонтников какие станки обслуживает.

**Табл. 7. О4**

|  |  |
| --- | --- |
| \* | \* + |
| Работает | На станке |
| Иванов А.В. | станок1 |
| Иванов А.В. | станок2 |
| Петров Н.Г. | станок1 |
| Петров Н.Г. | станок3 |
| Сидоров В.К. | станок2 |

**Табл. 8. Д4**

|  |  |
| --- | --- |
| \* | \* + |
| Обслуживает | Станок |
| Голубев Б.С. | станок1 |
| Голубев Б.С. | станок3 |
| Зыков А.Ф. | станок2 |
| Зыков А.Ф. | станок3 |

Первой и третьей записям таблицы О4 соответствует первая запись таблицы Д4 (у всех этих записей значения второго поля – «станок1»). Четвертой записи таблицы О4 соответствуют вторая и четвертая записи таблицы Д4 (во втором поле этих записей содержится «станок3»).

Исходя из определений полей связи этих таблиц можно составить новую таблицу с именем «О4+Д4», записями которой будут псевдозаписи. Записям полученной таблицы можно придать смысл возможных смен, составляемых при планировании работы. Для удобства, поля новой таблицы переименованы.

**Табл. 9. "О4+Д4"**

|  |  |  |
| --- | --- | --- |
| Работа | Станок | Обслуживание |
| Иванов А.В. | станок1 | Голубев Б.С. |
| Иванов А.В. | станок2 | Зыков А.Ф. |
| Петров Н.Г. | станок1 | Голубев Б.С. |
| Петров Н.Г. | станок3 | Голубев Б.С. |
| Петров Н.Г. | станок3 | Зыков А.Ф. |
| Сидоров В.К. | станок2 | Зыков А.Ф. |

Приведенную таблицу можно использовать, например, для получения ответа на вопрос: «Кто обслуживает станки, на которых трудится Петров Н.Г.?».

Очевидно, аналогично связи 1:1, связь М:М не устанавливает подчиненности таблица. Для проверки это можно основную и дополнительную таблицу поменять местами и выполнить объединение информации путем связывания. Результирующие таблицы «О4+Д4» и «Д4+О4» будут отличаться порядком следования первого и третьего полей, а также порядком расположения записей.

***Замечание.***

На практике в связь обычно вовлекается сразу несколько таблиц. При этом одна из таблиц может иметь различного рода связи с несколькими таблицами. В случаях, когда связанные таблицы, в свою очередь, имеют связи с другими таблицами, образуется иерархия или дерево связей.

#### Контроль целостности связей

Контроль целостности связей обычно означает анализ содержимого двух таблиц на соблюдение следующих правил:

* каждой записи основной таблицы соответствует нуль или более записей дополнительной таблицы;
* в дополнительной таблице нет записей, которые не имеют родительских записей в основной таблице;
* каждая запись дополнительной таблицы имеет только одну родительскую запись основной таблицы.

Опишем действие контроля целостности при манипулировании данными в таблицах. Рассмотрим три основные операции над данными двух таблиц:

* ввод новых записей,
* модификацию записей,
* удаление записей.

При рассмотрении попытаемся охватить все возможные методы организации контроля целостности. В реальных СУБД могут применяться собственные методы, подобные описываемым.

При вводе новых записей возникает вопрос определения последовательности ввода записей в таблицы такой, чтобы не допустить нарушения целостности. Исходя из приведенных правил, логичной является схема, при которой данные сначала вводятся в основную таблицу, а потом – в дополнительную. Очередность ввода может быть установлена на уровне целых таблиц или отдельных записей (случай одновременного ввода в несколько открытых таблиц).

В процессе заполнения основной таблицы контроль значений полей связи ведется как контроль обычного ключа (на совпадение со значениями тех же полей других записей). Заполнение полей связи дополнительной таблицы контролируется на предмет совпадения со значениями полей связи основной таблицы. Если вновь вводимое значение в поле связи дополнительной таблицы не совпадает ни с одним соответствующим значением в записях основной таблицы, то ввод такого значения должен блокироваться.

**Модификация записей**. Изменение содержимого полей связанных записей, не относящихся к полям связи, очевидно, должно происходить обычным образом. Нас будет интересовать механизм изменения полей связи.

При редактировании полей связи дополнительной таблицы очевидным требованием является то, чтобы новое значение поля связи совпадало с соответствующим значением какой-либо записи основной таблицы.

То есть дополнительная запись может сменить родителя, но остаться без него не должна.

Редактирование поля связи основной таблицы разумно подчинить одному из следующих правил:

* редактировать записи, у которых нет подчиненных записей. Если есть подчиненные записи, то блокировать модификацию полей связи;
* изменения в полях связи основной записи мгновенно передавать во все поля связи всех записей дополнительной таблицы (каскадное обновление).

В операциях удаления записей связанных таблиц большую свободу, очевидно, имеют записи дополнительной таблицы. Удаление их должно происходить практически бесконтрольно.

Удаление записей основной таблицы логично подчинить одному из следующих правил:

* удалять можно запись, которая не имеет подчиненных записей;
* запретить (блокировать) удаление записи при наличии подчиненных записей, либо удалять ее вместе со всеми подчиненными записями (каскадное удаление).

## **Основы проектирования баз данных**

В данном разделе приводятся основные принципы проектирования и разработки баз данных.

### *Целостность данных*

Способность гарантировать корректность извлеченных данных является первым принципом информационной архитектуры. И одновременно это – главная проблема, которая возникает в мире баз данных. При отсутствии целостности данных ответ на любой запрос может оказаться некорректным, следовательно, уже нет особого смысла в обеспечении производительности и доступности.

Поскольку данные представляют собой сущности и атрибуты, их целостность обеспечивается целостностью сущностей, домена, а также ссылочной и определяемой пользователем целостностью. Целостность транзакций, т.е. порядок записи и извлечения данных, определяется принципами ACID (атомарность, постоянство, изоляция и живучесть).

### *Целостность сущностей*

Целостность сущности имеет дело с ее структурой (первичным ключом и атрибутами). Если первичный ключ уникален, а все атрибуты скалярные и полностью зависят от первичного ключа, то целостность сущности обеспечена. В физической схеме целостность сущности поддерживает первичный ключ таблицы. Обеспечение целостности сущности непосредственно связано с нормализацией.

### *Целостность домена*

В терминах реляционной теории доменном называют множество возможных значений атрибута, будь то целочисленных, битовых или текстовых. Понятие целостности домена подразумевает, что в атрибуте пустых значений также входит в понятие целостности домена. В физической схеме основой поддержания целостности домена являются тип данных и допустимость пустых значений.

### *Ссылочная целостность*

Понятие ссылочной целостности, прежде всего, связано с целостностью домена внешних ключей. Целостность домена подразумевает, что если некоторый атрибут имеет значение, значит оно должно входить в домен. В случае внешних ключей доменом является список значений связанного первичного ключа.

В то же время ссылочная целостность является не вопросом целостности первичного ключа, а только связанного с ним внешнего.

Допустимость пустых значений в столбце – это отдельный вопрос, не связанный со ссылочной целостностью. Внешний ключ, в принципе, вполне может содержать пустые значения.

Некоторые методы поддержания ссылочной целостности реализуются на уровне физической схемы. В физической схеме может поддерживаться декларативная ссылочная целостность (DRI), и к таблице может быть прикреплен триггер.

### *Определенная пользователем целостность*

В стороне от требований целостности реляционной теории существует и определяемая пользователем целостность. Она может поддерживаться следующим образом:

* простыми бизнес-правилами, такими как ограничение на домен, т.е. ограничение списка допустимых сущностей данных. Для поддержания этих правил в физической схеме часто используют ограничения;
* сложными бизнес-правилами, которые ограничивают список допустимых данных на основе какого-либо условия. Например, некоторые туры требуют наличия у туристов определенных медицинских справок. Реализация этих правил в физической схеме обычно требует написания хранимых процедур или триггеров.

Некоторые требования целостности данных не могут быть обеспечены ограничениями или триггерами. Неполные, ошибочные или недостоверные данные могут успешно пройти все проверки и попасть в базу данных. Например, заказ без единой товарной строки считается некорректным, но ни один автоматический метод не сможет запретить ему попасть в базу. Запросы SQL могут выявить незаполненные заказы и помочь в других менее важных вопросах поддержания целостности данных, в том числе в следующих:

* поиск некорректных данных;
* поиск неполных данных;
* поиск сомнительных данных;
* поиск несогласованных данных.

Качество данных непосредственно зависит от людей, которые их вводят или изменяют. Защита данных (т.е. контроль над тем, кто именно может просматривать или заменять их) также является одним из аспектов поддержания их целостности.

### *Целостность транзакций*

*Транзакцией* называется единый логический блок работы, например, вставка 100 строк, обновление 1000 строк или выполнение логической цепочки обновлений. Качество продукта базы данных зависит от того, насколько его возможности выполнения транзакций соответствуют принципам ACID. Как вы помните, эта аббревиатура расшифровывается как четыре взаимозависимых свойства: атомарность, целостность, изоляция и живучесть.

* Свойство атомарности подразумевает, что транзакция должна либо выполняться вся, либо не выполняться в целом. В конце транзакции она должна быть либо подтверждена, либо отменена. Если частично выполненная транзакция записывается на диск, то свойство атомарности нарушается.
* Транзакция должна поддерживать целостность базы данных. Это значит, что транзакция начинает выполняться, когда база данных находится в целостном состоянии, при этом база должна остаться в целостном состоянии и после завершения транзакции. С точки зрения ACID целостность означает, что каждая строка и значение должны соответствовать моделируемой реальной ситуации, а все ограничения должны выполняться. Например, если заголовок заказа записан на диск, а строки заказанных в нем товаров – нет, то нарушается ссылочная целостность.
* Каждая транзакция должна быть изолирована, т.е. отделена от эффекта выполнения других транзакций. Независимо от того, что выполняет другая транзакция, первая должна иметь возможность продолжать работать с тем же набором данных, с которым начал. Изоляция является своеобразным забором между двумя транзакциями. Доказательством наличия изоляции является возможность многократно повторить последовательный набор транзакций над одним и тем же набором данных и каждый раз получить одни и тот же результат.
* Понятие живучести транзакции относится к ее выполнению независимо от системных сбоев. После того как транзакция будет подтверждена, она остается подтвержденной. СУБД должна быть спроектирована так, чтобы базу данных можно было восстановить до состояния последней полностью выполненной транзакции, даже если устройство данных даст сбой.

Узким местом целостности транзакций является конкуренция – попытка множества пользователей одновременно извлечь или модифицировать одни и те же данные. Вопрос изоляции реже встает в маленьких базах данных, однако в производственной среде с тысячами пользователей конкуренция соревнуется с целостностью транзакций. Эти два процесса должны быть сбалансированы, в противном случае целостность данных или производительности сильно пострадает.

### *Ошибки транзакций*

Понятие полной изоляции подразумевает, что одна транзакция никогда не помешает работе другой. Если изоляция полная, то все изменения, происходящие вне транзакции, она не увидит.

Изоляция между транзакциями может быть далеко не совершенной, что проявляется тремя способами: «грязное» чтение, неповторяющееся чтение и призрачные строки. К тому же транзакции могут завершаться неудачно из-за потерянных обновлений и взаимоблокировок.

#### «Грязное» чтение

Наиболее заметный сбой происходит, когда работа одной транзакции видна другой перед тем, как та подтвердит свои изменения. Когда одна транзакция может видеть неподтвержденные обновления другой, такая ситуация называется «грязным» чтением. Проблема «грязного» чтения состоит в том, что прочитанные данные еще не подтверждены, поэтому запись транзакцией данных может быть отменена.

#### Неповторяющееся чтение

Неповторяющееся чтение аналогично «грязному», однако оно происходит, когда одна транзакция в процессе работы видит подтвержденные в это время обновления другой транзакции. Чтении строки внутри транзакции всегда должно выдавать одни и те же результаты. Если при последовательных чтениях получаются разные результаты, значит, в транзакции произошла ошибка неповторяющегося чтения.

#### Призрачные строки

Наименее частой ошибкой являются призрачные строки. Подобно неповторяющемуся чтению, эта ошибка происходит. Когда обновления другой транзакции повлияли не только на результирующий набор данных, но и вызвали извлечение инструкцией SELECT отличного набора строк.

Среди всех ошибок «грязное» чтение можно назвать самым опасным, неповторяющееся чтение – немного менее опасным, и практически самыми безобидными являются призрачные строки.

#### Потерянные обновления

Потерянные обновления происходят, когда два пользователя редактируют одну и ту же строку, завершают свою работу и сохраняют данные. При этом обновления, которые были выполнены и сохранены первыми, заменяются обновлениями, сохраненными вторыми.

Так как потерянные обновления происходят только в том случае, когда два пользователя одновременно модифицируют одну и ту же строку, эта проблема может не возникать месяцами. Тем не менее это брешь в целостности транзакций, и эту ошибку в базах данных нужно не допускать.

#### Взаимоблокировки

Взаимоблокировкой называется особая ситуация, когда несколько задач транзакций соревнуются за одни и тот же ресурс данных. Для примера рассмотрим следующий сценарий.

* Первая транзакция блокировала данные A и собирается для своего завершения блокировать данные Б.
* В то же время вторая транзакция блокировала данные Б и собирается для своего завершения блокировать данные А.

Каждая из транзакций останавливается в ожидании, пока вторая освободит нужный ей ресурс. В результате ни одна из них не может завершить свою работу. Если извне не поспособствовать одной из них или если одна из транзакций не завершит свою работу, то эта ситуация может продолжаться до бесконечности.

#### Уровни изоляции

На физическом уровне любое ядро базы данных, допускающее логические транзакции, должно содержать механизмы изоляции. Для контроля, какие ошибки транзакций допустимы, существуют уровни изоляции. В спецификации ANSI SQL-92 определены четыре уровня изоляции:

* Read uncommitted;
* Read committed;
* Repeatable read;
* Serializable.

В дополнение к этому списку компания Microsoft в версии SQL Server 2005 добавила уровень изоляции Snapshot.

В SQL Server для установки уровня изоляции используется команда SET TRANSACTION ISOLATION LEVEL.

##### READ UNCOMMITTED

Указание READ UNCOMMITTED, по существу, эквивалентно использованию подсказки (hint) NOLOCK для каждой таблицы, задействованной в транзакции. Этот уровень наименее запретительных из пяти уровней изоляции SQL Server. Он позволяет осуществлять «грязное» чтение (чтение незафиксированных изменений другими транзакциями) и неповторяющееся чтение (то есть когда данные могут изменяться между чтениями в транзакции).

##### READ COMMITTED

READ COMMITTED – это уровень изоляции SQL Server по умолчанию, так что если вы не укажете другой, будет использован READ COMMITTED. READ COMMITTED не разрешает «грязное» чтение, накладывая коллективные блокировки (share locks) на данные, с которыми проводится работа, но позволяет изменять эти данные во время транзакции, что может привести к неповторяемому чтению и/или фантомным данным.

##### REPEATABLE READ

REPEATABLE READ накладывает блокировки, чтобы предотвратить изменение другими пользователями данных, с которыми работает транзакция, но не запрещает добавление новых записей, поэтому между чтениями в транзакции возможно возникновение фантомных записей.

##### SERIALIZABLE

SERIALIZABLE предотвращает «грязное» чтение и появление фантомных записей накладывая серию блокировок на данные. Это наиболее запретительный уровень из четырех уровней изоляции SQL Server. Указание SERIALIZABLE эквивалентно применению подсказки HOLDLOCK для каждой таблицы, используемой в транзакции.

##### SNAPSHOT

Обычно известен как «читатели не блокируют писателей, писатели не блокируют читателей». Этот уровень изоляции использует управление версиями строк и гарантирует, что операции чтения возвращают образ данных в их первоначальном виде до начала операций изменения. В сущности, этот уровень создает виртуальную копию, или мгновенный снимок, данных первой транзакции, в результате чего остальные транзакции на нее не влияют. Этот метод может привести к потерянным обновлениям.

### *Пустые значения*

Реляционные базы данных всегда представляют отсутствие данных с помощью специального пустого значения null. На самом деле null представляет три совершенно различных сценария отсутствия данных.

* Столбец не задействован в данной строке. Например, если человек не устроен на работу, значит, какое-либо определенной значение в столбце даты приема на работу будет некорректным.
* Данные еще не введены, но вскоре это будет сделано. Например, если контактное лицо имеет имя и номер телефона, а адрес будет введен в процессе приема заказа.
* Столбец в данной строке вообще не содержит значения. Например, столбец комментария заполняется не для всех сотрудников, представленных в списке.

В зависимости от типа отсутствующих данных некоторые проектировщики баз данных по умолчанию подставляют нули, пустые строки и т.п. Однако такие значения могут негативно отразиться на результатах некоторых запросов и привести к проблемам поддержания целостности данных.

### *Индексация*

Индексы являются своеобразным мостом между запросом и данными. Они повышают производительность за счет сокращения числа считывания страниц, на которых размещена таблица.

* Группа строк таблицы с кластеризованным индексом может быть считана за одно (или несколько) обращение к физическим страницам; при этом не приходится постоянно перескакивать по разным страницам.
* Индексы позволяют оптимизатору запросов напрямую переходить к конкретным строкам данных, а не сканировать для этого всю таблицу. Это аналогично тому, как предметный указатель книги помогает найти нужную страницу. Как только нужная строка найдена, оптимизатор выполняет операцию поиска закладки для перехода к конкретной странице.

Индексация является ключевым моментом физической схемы.

Однако индексы имеют и отрицательную сторону. Несмотря на то что они помогают осуществлять поиск по таблице, они могут негативно влиять на скорость записи. Когда вставляется или обновляется строка, индексам может потребоваться провести и синхронизацию с данными. Таким образом, когда таблица имеет множество индексов, запись в таблицу будет замедляться. Другими словами, существует разногласие между созданием индексов для чтения и записью. Так как операции обновления и удаления должны найти измененные строки, операции записи дадут преимущества при экономной и обдуманной индексации.

## **Введение в Microsoft SQL Server 2005**

SQL Server – нечто большое, чем просто ядро реляционной базы данных. Это набор служб и компонентов, связанных с базами данных, которые могут совместно использоваться для построения мощных решений для конечного пользователя.

### *Службы SQL Server*

### *Реляционное ядро*

Реляционное ядро SQL Server 2005, иногда называемое ядром базы данных, является сердцем SQL Server. Это процесс, поддерживающий всю реляционную работу базы данных. SQL – декларативный язык, т.е. он описывает ядру базы данных запрос, который тот должен выполнить. Именно он заставляет работать этот механизм.

SQL Server поддерживает установку до пятидесяти экземпляров реляционного ядра на одном физическом сервере. Несмотря на то что все установленные экземпляры совместно используют некоторые компоненты, каждый экземпляр функционирует как полноценная установка SQL Server.

В реляционном ядре существует несколько ключевых процессов и компонентов, в том числе следующие.

* **Оптимизатор запросов (Query Optimizer).** Определяет, как обрабатывать запрос, основываясь на относительных затратах, которые влечет за собой выполнение определенных типов операциий. Оцененный и фактический планы выполнения запроса можно просмотреть в графическом представлении или в форме XML с помощью утилиты Management Studio (SSMS).
* **Диспетчер буфера (Buffer Manager).** Анализирует используемые страницы данных и предварительно загружает определенные фрагементы файлов данных в память, снижая таким образом зависимость от производительности операций ввод-вывода на диск.
* **Lazy Writer.** Записывает страницы данных, которые были изменены в памяти, в файл данных.
* **Монитор ресурсов (Resource Monitor).** Оптимизирует кэш планов запросов в соответсвии с нагрузкой на память и избирательно удаляет из него старые планы.
* **Диспетчер блокировок (Lock Manager).** Динамически управляет множеством блокировок, выполняя балансировку количества затребованных блокировок с их размерами.
* SQL Server поглощает ресурсы компьютера, поэтому ему нужен прямой контроль за доступными русурсами (памятью, потоками, запросами на ввод-вывод и т.п.). Отдать управление ресурсами на откуп операционной системе совсем не сложно для SQL Sever 2005. Она содержит собственный уровень ОС, называемый SQLOS, который управляет всеми внутреними ресурсами.

### *Transact-SQL*

СУБД SQL Server основана на стандарте SQL с некоторыми добавленными компанией Microsoft расширениями. Стандарт SQL впервые был введен доктором Коддом в 1971 году, когда он работал в исследовательской лаборатории IBM в Сан-Хосе. СУБД SQL Server 2005 в своей основе совместима со стандартом ANSI SQL-92.

Хотя стандарт ANSI SQL можно назвать идеальным для команд общего назначения отбора и определения данных, в него не входят, команды предназначенные для управления свойствами SQL Server и логического управления пакетами, необходимые для создания специфичных для приложений SQL Server. Исходя из этого, команда Microsoft SQL Server расширила спецификацию ANSI SQL рядом новых команд, а некоторые команды реализовала несколько по своему. Результатом стал язык Transact-SQL, или, сокращенно, T-SQL, - диалект SQL, распознаваемый SQL Server.

В T-SQL нет некоторых команд ANSI SQL (таких, как определение каскадности внешних ключей, обработки пустых значений и умолчаний) в основном потому, что компания Microsoft реализовала эти функции иначе. По умолчанию T-SQL несколько по-иному, чем требует стандарт ANSI, обрабатывает пустые значения, кавычки и заполнения, но этим режимом можно управлять.

Чтобы освоить SQL Server, нужно ознакомить с языком T-SQL, поскольку ядро SQL Server понимает только это язык. Любая команда, посылаемая серверу, должна поддерживать правила этого языка. Пакеты хранимых команд T-SQL могут выполняться на сервере как хранимые процедуры. Другие инструменты, такие как Management Studio, реализующие графический интерфейс пользователя, позволяют управлять сервером, преобразуя щелчки мышью на определенных инструментах в соответствующие команды T-SQL, отправляемые ядру базы данных.

Команды SQL и T-SQL можно разбить на три категории.

* **Язык манипулирования данными (Data Manipulation Language - DML).** Включает в себя популярные инструкции SQL: select, insert, update и delete. DML иногда ошибочно называют языком модификации данных (Data Modification Language), но это неверно, поскольку инструкция select данные не модифицирует, а извлекает.
* **Язык определения данных (Data Definition Language - DDL).** Содержит инструкции, создающие и модифицирующие таблицы данных, ограничения и прочие объекты базы данных.
* **Язык управления данными (Data Control Language - DCL).** Содержит инструкции управления системой безопасности, такие как grant, revoke и deny.

### *Visual Studio и CLR*

Одним из самых впечатляющих новшеств является возможность совместной работы систем Visual Studio и SQL Server 2005. Несмотря на то что и в прошлом эти системы находились в тесной связи, сейчас интеграция Visual Studio 2005 и SQL Server 2005 сильнее, чем когда-либо раньше. Эти два продукта разрабатывались и были выведены на рынок одновременно, и это сразу видно. Утилита Management Studio основана на интегрированной среде разработки (IDE) пакета Visual Studio. Однако интеграция сильнее, чем на первый взгляд видно по утилите Management Studio. Внутренняя операционная система пакета SQL Server 2005, SQLOS, на самом деле управляет общеязыковым интерпретатором .NET (CLR) внутри SQL Server.

Сборки, созданные в Visual Studio, могут быть развернуты и запущены в SQL Server в качестве хранимых процедур, триггеров, определенных пользователем обычных или агрегатных функций. К тому же типы данных, созданные в Visual Studio, могут использоваться для определения таблиц и хранения дополнительных данных.

Достаточно важно то, что в SQLOS поддерживается CLR. Это значит, что SQL Server управляет ресурсами CLR. СУБД может разрешить проблемы CLR, завершить выполнение и перезапустить процедуру CLR, которая стала источником проблем, и гарантировать, что в вечной войне за память победит нужный игрок.

### *Служба репликаций*

Данные SQL Server часто требуются в пространстве трансконтинентальных корпораций и прочих глобальных организаций, и при этом для перемещения данных часто используют репликации SQL Server. Дополненная новыми инструментами, служба репликаций SQL Server 2005 может перемещать транзакции в одностороннем направлении, а также объединять обновления из многих мест с помощью топологии издателя, распространителя и подписчика.

### *Брокер служб*

Впервые появившийся в версии SQL Server 2005 брокер служб (Service Broker) управляет асинхронными очередями данных, поддерживая ключевые средства производительности и масштабируемости и балансируя загрузку в течении времени.

* Брокер служб может буферизовать большие объемы вызовов Web-служб HTTP или хранимых процедур. Вместо того чтобы вызов тысячи Web-служб запустил одновременно на выполнение тысячу потоков хранимых процедур, эти вызовы помещаются в очередь, и хранимые процедуры могут быть вызваны несколькими экземплярами SQL Server, что более эффективно распределяет нагрузку.
* Серверные процессы, содержащие сложную логику или периоды сильной нагрузки могут поместить необходимые данные в очередь и вернуть их вызывающему процессу без нарушения логики. Брокер служб может провести через очередь вызов другой хранимой процедуры, которая возьмет на себя основную нагрузку.

Несмотря на то, что в SQL Server можно создать собственные очереди, существует ряд преимуществ в использовании стандартных рабочих очередей Microsoft. SQL Server содержит команды DDL, управляющие брокером служб; есть команды T-SQL, помещающие данные в очередь и извлекающие их из нее. Информация об очередях брокера служб представлена в метаданных представлений, а также в утилитах Management Studio и System Monitor. Наиболее важно то, что брокер служб основательно протестирован и спроектирован специально для систем с исключительно большой нагрузкой.

### *Полнотекстовый поиск*

Полнотекстовый поиск существовал в SQL Server с версии 7, но в каждой следующей версии этот прекрасный инструмент улучшался. Для быстрого поиска строк запросы SQL Server используют индексы. По умолчанию индекс строиться по всему столбцу. Поиск отдельных слов внутри столбцов представляет собой очень медленный процесс. Полнотекстовый поиск (Full-Text Search) решил эту проблему, индексируя каждое из слов в столбце.

Когда полнотекстовый поиск был разработан для столбцов, запросы SQL Server получили возможность использовать при поиске полнотекстовые индексы и быстро находить в строках отдельные слова.

Служба Microsoft Search Service, выполняющая полнотекстовый поиск, на самом деле является компонентом операционной системы и предназначена для поиска текста в файлах. SQL Server при выполнении полнотекстового поиска берет на вооружение именно эту службу. Эта служба может быть остановлена и запущена в утилите SQL Server Configuration Manager или в консоли Службы панели управления Windows.

### *Служба уведомлений*

Изначально выпущенная в 2003 году в качестве надстройки для SQL Server 2000, служба уведомлений (Notification Service) может быть запрограммирована для отправки сообщений при возникновении определенных событий с данными. Сообщения могут отсылаться практически на любое устройство, в том числе на пейджеры, мобильные телефоны и по электронной почте. Типичным приложением службы уведомлений может быть система сообщений на транспортных узлах, которая автоматически информирует пассажиров об изменении в расписании и задержках рейсов.

### *SQL Server Agent*

SQL Server Agent представляет собой дополнительный процесс, который выполняет задания SQL и обслуживает прочие автоматизированные задачи. Он может быть сконфигурирован для автоматического запуска при загрузке системы; также он может запускаться вручную из утилит SQL Server Configuration Manager и SQL Server Surface Area Configuration Tool.

### *Координатор распределенных транзакций*

Координатор распределенных транзакций (Distributed Transaction Coordinator - DTC) представляет собой процесс, поддерживающий двухфазные подтверждения для транзакций, которые охватывают несколько серверов. Процесс DTC может быть запущен из консоли служб панели управления Windows. Если приложение регулярно использует распределенные транзакции, то лучше сконфигурировать DTC для автоматического запуска при загрузке операционной системы.

### *SQL Mail*

Компонент SQL Mail позволяет серверу отправлять электронные сообщения на внешний почтовый ящик по протоколу SMTP. Почтовые сообщения могут генерироваться множеством источников в SQL Server, в том числе кодом T-SQL, заданиями, предупреждениями, службой интеграции и планами обслуживания.

### *Службы бизнес-аналитики*

Бизнес-аналитика (Business Intelligence – BI) – это одна из областей, в которой SQL Server 2005 превосходит все остальные системы. Развитие средств бизнес-аналитики в SQL Server за последние несколько лет было стремительным. Результатом стало то, что версия SQL Server 2005 включает в себя три службы, специально предназначенные для бизнес-аналитики: службу интеграции (иногда для нее используется аббревиатура SSIS), службу отчетности (RS) и службу анализа (AS). Все эти три службы были созданы с помощью среды BI Development Studio и управляются из утилиты Management Studio.

### *Служба интеграции*

Служба интеграции (Integration Service) перемещает данные практически между любыми типами источников. Она является инструментом извлечения, преобразования и загрузки данных СУБД SQL Server.

Пакеты службы интеграции достаточно гибкие, что как напрямую копировать данные из столбца в столбец, так и выполнять сложные преобразования, использовать классификаторы и обработку исключений во время перемещения данных, сборе информации из разнородных источников и обобщении информации хранилищ данных для анализа службой Analysis Services. Эта служба даже имеет инструменты для работы с нечеткими множествами данных.

### *Служба отчетности*

Служба отчетности (Reporting Services) в SQL Server 2005 представляет собой полнофункциональное, основанное на Web-технологиях, управляемое решение. Ее отчеты могут быть экспортированы в форматы PDF, Excel и многие другие с помощью одного щелчка мышью; их также легко создавать и настраивать.

Отчеты определяются графически или программно и хранятся в качестве файлов .rd в базе данных служб отчетности в SQL Server. Их форматированию может быть назначено расписание, они могут кэшироваться для пользователей, отправляться им по электронной почте или генерироваться по запросу с задаваемыми пользователями параметрами. Служба отчетности встроена в SQL Server.

### *Служба анализа*

Служба анализа (Analysis Services) решает в SQL Server задачи бизнес-аналитики, или, точнее, оперативной аналитической обработки данных (OLAP). В сущности, служба анализа позволяет разработчику определять кубы, которые в чем-то сходны со сводными таблицами Excel и перекрестными запросами Access, но способные иметь множество измерений. Кубы содержат предварительно рассчитанные сводные или консолидированные данные из особо крупных баз данных. Это позволяет пользователю просто и быстро просматривать эти итоги без необходимости выполнения «долгоиграющих» запросов к терабайтам данных.

Служба анализа загружается независимо от SQL Server и рассматривается как одна из полноценных функций формирования хранилища данных.

### *Различные редакции SQL Server 2005*

Пакет SQL Server 2005 доступен в нескольких редакциях, которые отличаются своими возможностями и требованиями к аппаратному обеспечению.

#### Enterprise (Developer) Edition

Это самая полная редакция продукта с повышенной производительностью и расширенным набором функций. Она способна поддерживать тысячи подключений и базы данных, измеряемые терабайтами. Для достижения такого уровня производительности требуются и соответствующие компьютеры – как минимум, с шестнадцатью двухъядерными процессорами, 32 Гбайт памяти и мощными сетевыми платами.

Чтобы получить полную отдачу от такой аппаратуры, редакция Enterprise Edition вобрала в себя некоторые прекрасные функции. Две из них стоит упомянуть отдельно.

* Разделение таблиц. Эта функция позволяет получить удивительные результаты производительности при использовании протокола TCP.
* Параллельное индексирование в реальном времени.

Редакция для разработчиков Developer Edition имеет тот же состав, что и Enterprise Edition, с двумя исключениями. Во-первых, Developer Edition лицензируется только для разработки приложений и их тестирования, так что легально использовать эту редакцию в производственной среде нельзя. Во-вторых, Developer Edition запускается в операционных системах, предназначенных для рабочих станций, например Windows NT Workstation, Windows 2000 Professional и Windows XP Professional.

#### Standard Edition

Потребности большей части средних и крупных производственных баз данных полностью удовлетворяет редакция Standard Edition. Это самая популярная редакция продукта, которая включает в себя все основные функции, в том числе службы интеграции и анализа, Web-службы, зеркальное отображение баз данных и кластеризацию.

Standard Edition отлично подходит для предприятий среднего размера, которым нужны все предлагаемые сервером функции, но не требуется сверхвысокая доступность Enterprise Edition. Эта редакция лимитирована четырехпроцессорными компьютерами, независимо от количества ядер в процессорах, но в то же время не налагает никаких ограничений на память.

Workgroup Edition

Эта редакция предназначена для серверов подразделений компаний и содержит рациональный набор функций для небольших транзакционных баз данных.

* Ядро базы данных поддерживает два процессора, максимум 3 Гбайт памяти и не имеет ограничений на размер базы данных.
* Поддерживаются некоторые функции высокой доступности, такие как доставка журналов, но не поддерживается зеркальное отображение баз данных и кластеризация.
* Включает службу отчетности, но не включает службу анализа для формирования кубов и бизнес-анализа.
* Имеет все программные возможности SQL Server 2005, включая T-SQL, хранимые процедуры, определяемые пользователем функции, полнотекстовый поиск, XML, XQuery, брокер служб и интеграцию .NET. Не поддерживаются Web-службы и служба уведомлений.
* Поддерживается администрирование автоматизации с помощью SQL Server Agent.
* Поддерживаются репликации транзакций и слияния.

Ключевой функцией, отсутствующей в это редакции, является служба интеграции. При этом предполагалось, что база данных для рабочей группы скорее будет являтся источником данных для других, более крупных серверов, но не будет содержать подчиненных источников данных.

#### Express Edition

Express Edition нельзя рассматривать как замену ядра Access Jet. Это бесплатная, полноценная версия ядра SQL Server, предназначенная для использования с некоторым приложением. Редакция Express имеет некоторые ограничения: максимальный объем базы данных составляет 4 Гбайт, поддерживается только один процессор и 1 Гбайт памяти. В настоящее время в этой редакции недоступна служба отчетности, однако компания Microsoft обещает включить ее в недалеком будущем.

#### Everywhere Edition

Редакция SQL Server Everywhere, изначально называвшаяся Mobile Edition, с технической стороны представляет собой полностью отличное ядро базы данных, совместимое с SQL Server. Ее скромные потребности в памяти (всего 1 Мбайт) гарантирует ее использование на мобильных устройствах. Несмотря ан то что она запускается на карманных компьютерах, это полноценное ACID-совместимое ядро базы данных.

### *Утилиты и компоненты SQL Server*

Для управления SQL Server и взаимодействия с ней используются следующие компоненты и клиентские утилиты.

#### SQL Server Management Studio

По сути Management Studio представляет собой Visual Studio – специализированную интегрированную среду, используемую администраторами и разработчиками баз данных. Ядром этой утилиты является Object Explorer, укомплектованный фильтрами и способный обозревать все серверы в составе SQL Server (ядро базы данных, сервер анализа, сервер отчетности и т.д.). Редактор запросов (Query Editor) этой утилиты является удобным механизмом работы с кодом T-SQL; он интегрирован с обозревателем решений (Solution Explorer) для управления проектами. Несмотря на то, что интерфейс этой утилиты кажется перегруженным все окна легко конфигурируются и могут быть автоматически скрыты с экрана.

#### SQL Server Configuration Manager

Этот инструмент используется для запуска и остановки любого сервер, настройки параметров запуска и конфигурирования подключений. Эту утилиту можно вызвать из системного меню Пуск, а также из Management Studio.

#### SQL Profiler

Утилита SQL Profiler наблюдает за всеми событиями и пакетами SQL Server, выводя выбранную информацию на экран, записывая в таблицу или файл. Эта утилита идеально подходит для отладки приложений и настройки базы данных. Функцию Database Tuning Advisor можно использовать для сбора данных с целью оптимизации базы данных.

#### Perfomance Monitor

В то время как SQL Profiler записывает большие наборы данных, касающиеся потоков SQL и событий SQL Server, утилита Perfomance Monitor (или System Monitor) выводит в открытое окно текущее состояние выбранных счетчиков. Утилиту Perfomance Monitor можно найти в папке администрирования панели управления системы Windows. Если на компьютере установлен пакет SQL Server, то его счетчики автоматически добавляются в Perfomance Monitor.

#### Database Tuning Advisor

Утилита Database Tuning Advisor анализирует пакет запросов (полученный из утилиты Profiler) и рекомендует изменить структуру индексов и разделов для повышения производительности. Этот пакет изменений можно легко сконфигурировать и применить полностью или частично либо сразу после анализа, либо позже.

#### Утилиты командной строки: SQLCmd и BulkCopy

Эти интерфейсы командной строки позволяют разработчику выполнить инструкции SQL или операции массового копирования из операционной системы DOS или командной строки Windows. Служба интеграции и SQL Server Agent делают эти утилиты устаревшим наследием прошлого, однако для обеспечения максимальной гибкости компания Microsoft решила включить их в состав SQL Server.

### *Обзор метаданных*

После изначальной установки SQL Server уже содержит некоторые системные и пользовательские объекты. Для самообслуживания SQL Server использует четыре системные базы данных и две базы данных примеров для экспериментов пользователей. При этом каждая из баз данных содержит несколько системных объектов, включая таблицы, представления, хранимые процедуры и функции.

В утилите Management Studio эти системные объекты могут быть скрыты. На странице Registered SQL Server Properties вы можете выбрать, что можно отображать, с помощью параметра Show system database and system objects.

#### Системные базы данных

SQL Server использует четыре системные базы данных для хранения системной информации, отслеживания операций и поддержки временной рабочей области. В дополнение база данных моделей может служить прототипом для новых пользовательских баз. Пять системных баз данных описаны ниже.

* **Master**. Содержит информацию обо всех базах данных, размещенных на сервере. К тому же объекты базы Master доступны и для остальных баз данных. Например, хранимые процедуры базы данных Master можно вызывать из любых пользовательских баз.
* **MSDB**. Содержит список действий, таких как резервное копирование и выполнение заданий, а также определяет порядок резервного копирования всех пользовательских баз данных.
* **Tempdb**. Используется для хранения временных таблиц, создаваемых пользователями, пакетами, хранимыми процедурами (включая системные) и сами ядром SQL Server. Если серверу необходимо создать временные кучи или списки в ходе выполнения запроса, то все они создаются именно в базе данных Tempdb. Эта база данных полностью очищается при перезапуске SQL Server.
* **Reference**. Эта скрытая база данных является разделом базы данных Master и размещена в том же каталоге. Она предназначена для облегчения установки пакетов обновлений.

#### Представления метаданных

Метаданными называют общую информацию о самих данных. Одним из исходных правил доктора Кодда для реляционных баз данных было хранение информации о базе данных в самой базе с помощью таблиц, строк и столбцов, что ничем не отличается от хранения данных пользователя. Эта информация о данных облегчает программную навигацию по схеме базы и ее конфигурации. SQL Server 2005 имеет несколько типов метаданных: представление каталогов, представления динамического управления и два представления системных таблиц.

* **Представления каталогов** содержит информацию о статистических метаданных, таких как таблицы, а также конфигурации системы безопасности и самого сервера.
* **Динамические представления управления и функции** позволяют заглянуть в текущее состояние сервера в данную секунду и увидеть информацию о таких характеристиках, как память, потоки, хранимые процедуры в кэше и подключения.
* **Представления совместимости** позволяют имитировать системные таблицы такими, какими они были в предыдущих версиях SQL Server.
* **Представления информационной схемы** не поддерживаются стандартом ANSI SQL-92 и служат для обзора схемы любого продукта базы данных. Это представление используется в крайнем случае и имеет малое практическое применение у администраторов баз данных и разработчиков, которые исследуют возможности SQL Server.

## **Программирование на языке Transact-SQL**

Стандартные инструкции языка манипулирования данными DML предназначены только для извлечения и изменения данных. Этот язык лишен средств разработки процедур и алгоритмов, а также команд настройки сервера и управления им. В качестве компенсации любая конкретная реализация языка SQL дополняет его своими расширениями.

Язык Transact-SQL, больше известный как T-SQL, является реализацией стандарта SQL, дополненной набором расширений Microsoft. Основной задачей языка T-SQL является обеспечение программиста набором средств проектирования транзакционных баз данных.

Язык T-SQL часто ассоциируют с хранимыми процедурами, но на самом деле он представляет нечто большее. В приложениях «клиент/сервер» он может быть задействован несколькими способами.

* T-SQL может использоваться в выражениях как часть команд DML: INSERT, UPDATE и DELETE, порожденных клиентским процессом.
* T-SQL может использоваться в программных блоках, направляемых от клиента к серверу.
* Функции T-SQL могут использоваться в качестве выражений в ограничениях проверки.
* Код T-SQL может использоваться в хранимых процедурах, функциях и триггерах сервера баз данных.

### *Основы Transact-SQL*

Язык T-SQL предназначен для управления наборами данных. По этой причине он не обладает некоторыми характерными чертами традиционных языков, которые необходимы для программирования приложений.

#### Пакеты T-SQL

Запросом называют одну инструкцию T-SQL, а пакетом – их набор. Вся последовательность инструкций пакета отправляется серверу из клиентских приложений как одна цельная единица.

SQL Server рассматривает весь пакет как рабочую единицу. Наличие ошибки хотя бы в одной инструкции приведет к невозможности выполнения всего пакета. В то же время грамматический разбор не проверяет имена объектов и схем, так как сама схема может измениться в процессе выполнения инструкции.

#### Прерывание выполнения пакета

Файл сценария SQL и окно анализатора запросов (Query Analyzer) может содержать несколько пакетов. В данном случае все пакеты разделяют ключевые слова терминаторов. По умолчанию этим ключевым словом является GO, и оно должно быть единственным в строке. Все другие символы (даже комментарии) нейтрализуют разделитель пакета.

Разделитель пакетов на самом деле является функцией Management Studio, а не самого сервера.

#### Инструкции DDL

Некоторые инструкции DDL языка T-SQL, такие как Create Procedure, обязательно должны быть первыми инструкциями пакета. Очень длинные сценарии, которые создают множество объектов, часто требуют наличия нескольких разделителей пакетов. Так как SQL Server отдельно разбирает синтаксис по пакетам, такое наличие множества разделителей помогает локализовать ошибки.

#### Переключение между базами данных

В интерактивном режиме работы текущая база данных всегда отображается на панели инструментов, и в любой момент может быть изменена. В программном коде текущая база определяется с помощью ключевого слова USE. Это ключевое слово в пакете указывает, с какой именно базой данных будет выполняться работа, начиная с текущей точки:

USE CHA2

На практике рекомендуется явно определять текущую базу данных с помощью команды USE.

#### Выполнение пакетов

Пакет может быть выполнен несколькими способами.

* Сценарий SQL в полном объеме (т.е. все входящие в него пакеты) может быть выполнен путем открытия файла .sql в редакторе SQL утилиты Management Studio и нажатия клавиши <F5> (или щелчка на кнопке !Execute панели инструментов, или выбор в меню пункта Query->Execute).
* В приложении пакет T-SQL можно выполнить с помощью ADO или ODBC.
* Сценарий T-SQL может быть выполнен с помощью утилиты командной строки SQLCmd с передачей ей имени файла .sql в качестве параметра.
* Утилита SQLCmd имеет несколько параметров и может быть легко сконфигурирована практически для любых нужд.

### *Реализация физической схемы базы данных*

#### Проектирование физической схемы базы данных

При проектировании физической схемы следует начинать с чисто логической модели – хорошо разобраться в основных бизнес-правилах и задокументировать их.

Реализация физической схемы базы данных включает в себя шесть компонентов.

* Создание файлов базы данных.
* Создание таблиц.
* Создание первичных и внешних ключей.
* Создание столбцов данных.
* Создание ограничений, гарантирующих целостность данных.
* Создание индексов (в принципе индексы могут быть без труда созданы и изменены уже после реализации физической схемы).

Преобразование логической схемы базы данных в физическую может включать несколько этапов.

* Преобразование сложного логического проекта в более простые и гибкие структуры таблиц.
* Преобразование составных первичных ключей в опирающихся только на один столбец.
* Преобразование бизнес-логики в ограничение и триггеры.
* Преобразование логических отношений «многие ко многим» в пары физических отношений «один ко многим», использующих ассоциативные (или связывающие) таблицы.

#### Создание базы данных

База данных является физическим контейнером для всех схем данных, а также для серверных программ. База данных SQL Server представляет собой одну логическую единицу, даже несмотря на то, что физически может размещаться в нескольких файлах.

#### Команда DDL CREATE

Операция создания базы данных с использованием параметров, заданных по умолчанию, крайне проста. Например:

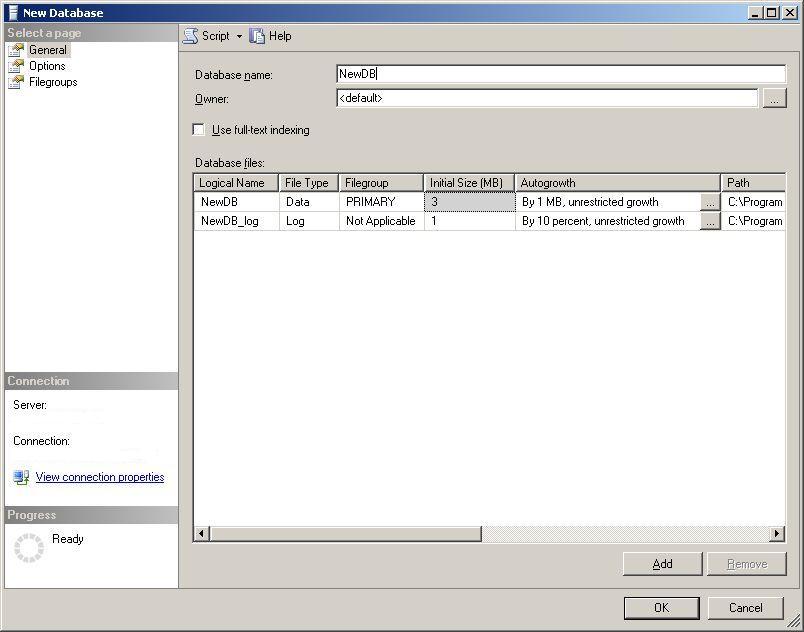
CREATE DATABASE CHA2

Инструкция CREATE создает файл данных с заданным именем и расширением .mdf, а также файл журнала транзакций с расширением .ldf.

Естественно, команда CREATE поддерживает множество параметров. По умолчанию приняты следующие.

* Порядок символов по умолчанию – принятый на сервере.
* Исходный размер. С помощью инструкции создается файл объемом порядка 2 Мбайт; с помощью Object Explorer создается файл объемом 3 Мбайт. Файл журнала транзакций при использовании кода имеет исходный объем 0,5 Мбайт; при создании в Object Explorer – 1 Мбайт.
* Размещение. Файлы данных и журнала транзакций по умолчанию размещаются в каталоге, принятом в SQL Server по умолчанию.

Создание баз данных в окне Object Explorer требует только ввода ее имени в форме, показанной на Рис. 1. Данное окно открывается после щелчка правой кнопки мыши на узле Database и выбора в контекстном меню пункта New Database.



**Рис. 3. Форма свойств новой базы данных.**

Окно Database Properties состоит из нескольких страниц: General, Options и Filegroups. Для существующих баз данных в это окно добавляются страницы Files, Permissions, Extended Properties, Mirroring и Log Shipping. Все страницы окна Database Properties описаны в Табл. 1.

**Табл. 10. Страница окна свойств базы данных**

|  |  |  |
| --- | --- | --- |
| **Страница** | **Новая база данных** | **Существующая база данных** |
| General | Создание новой базы данных, настройка имени, владельца, упорядочения, модели восстановления, полнотекстовой индексации и свойств файла данных. | Просмотр (только чтение) общих свойств: имени, даты последнего резервирования, размера и упорядочения. |
| Files | Нет данных. | Просмотр и изменение владельца базы данных, упорядочения, модели восстановления, полнотекстовой индексации и файлов базы данных. |
| Filegroups | Просмотр и изменение информации о файловых группах. | Просмотр информации о файловых группах. |
| Options | Просмотр и модификация параметров базы данных, таких как автосжатие, настройки ANSI, метод верификации страниц и монопольного доступа | Просмотр и модификация параметров базы данных, таких как автосжатие, настройки ANSI, метод верификации страниц и монопольного доступа |
| Permissions | Нет данных | Просмотр и модификация серверных ролей, пользователей и разрешений. |
| Extended Properties | Нет данных | Просмотр и модификация расширенных свойств |
| Mirroring | Нет данных | Просмотр и конфигурирование зеркального отображения базы данных |
| Transaction Log Shipping | Нет данных | Настройка доставки журнала транзакций |

#### Концепции файлов базы данных

База данных состоит из двух файлов (точнее, из двух наборов файлов): файла данных и журнала транзакций. Файл данных содержит системные и пользовательские таблицы, индексы, представления, хранимые процедуры, пользовательские функции, триггеры и разрешения системы безопасности. Журнал транзакций с последовательной записью является «центром» SQL Server. Вcе обновления данных вначале записываются в журнал транзакций и проверяются, что гарантирует запись обновлений в двух местах.

Журнал транзакций содержит не только пользовательские записи, но и индексные записи, разбиения страниц, реорганизации таблиц и многое другое.

#### Использование множества файлов

Как данные, так и журнал транзакций могут храниться во множестве файлов. Это позволит увеличить быстродействие и открыть новые перспективы для увеличения объема файлов. Любой дополнительный (или вторичный) файл данных имеет по умолчанию расширение .ndf. Если база использует множество файлов данных, то системные таблицы будут храниться в первом из них (т.е. первичном).

#### Создание базы данных с несколькими файлами

Для того чтобы создать базу данных с несколькими файлами в Management Studio, добавляйте имена файлов в таблицу Database files на странице Files диалогового окна параметров базы данных (Рис. 2).

При программном создании базы данных с несколькими файлами добавьте их место размещения в инструкцию CREATE DATABASE.

CREATE DATABASE NewDB

ON

PRIMARY

(NAME = NewDB,

FILENAME = ‘e:\SQLData\NewDB.mdf’),

(NAME = NewDB2,

FILENAME = ‘f:\SQLData\NewDB2.ndf’),

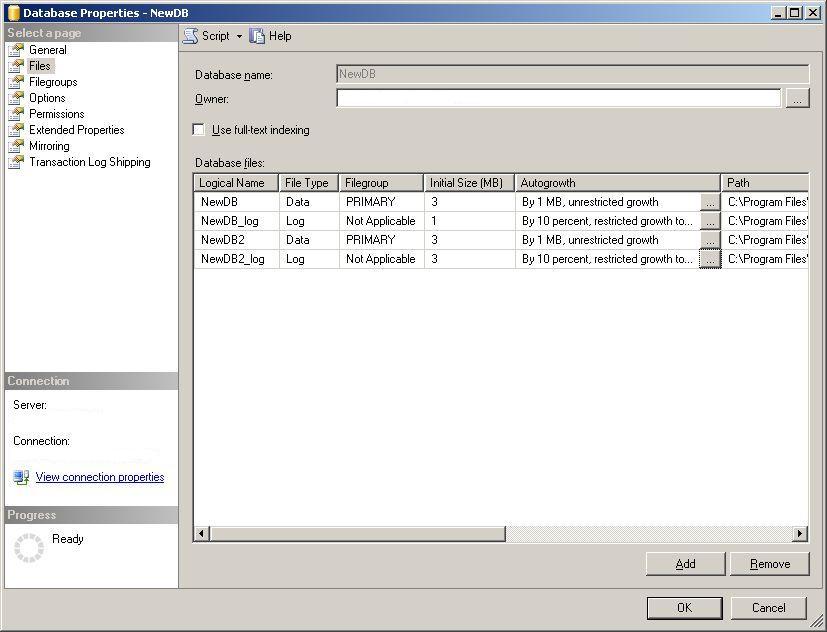
LOG ON

(NAME = NewDBLog,

FILENAME = ‘g:\SQLLog\NewDBLog.ldf’),

(NAME = NewDBLog2,

FILENAME = ‘h:\SQLLog\NewDBLog2.ldf’);



**Рис. 4. Создание базы данных с несколькими файлами в Management Studio**

Изменение файлов текущей базы данных

Количество файлов, используемых текущей базой данных, можно легко изменить. Если данные заполнили устройство, вы можете добавить новый файл в таблице Database Files (Рис. 2) так же, как и при создании базы данных.

В программном коде можно выполнить ту же операцию с помощью инструкции ALTER DATABASE с параметром ADD FILE. Синтаксис этой инструкции аналогичен использовавшейся при создании базы данных. В следующем примере к базе данных NewDB добавляется новый файл:

ALTER DATABASE NewDB

ADD FILE

(NAME = NewDB3,

FILENAME = ‘i:\SQLData\NewDB3.ndf’,

SIZE = 10 MB,

MAXSIZE = 2 Gb,

FILEGROWTH = 20);

Если некоторый файл больше не требуется либо по причине вывода из эксплуатации дисковой подсистемы, либо из-за ее использования для других целей, файл данных или журнала транзакций может быть удален. Для этого его нужно вначале сжать с помощью функции DBCC ShrinkFile, а затем удалить в утилите Management Studio (для этого нужно выделить его и нажать клавишу <Delete>).

Дополнительные файлы можно также удалить и программным путем с помощью инструкции ALTER DATABASE REMOVE FILE. Следующая инструкция удаляет созданный ранее третий файл базы данных NewDB:

DBCC SHRINKFILE (NewDB3, EMPTYFILE)

ALTER DATABASE NewDB

REMOVE FILE NewDB3;

#### Планирование нескольких файловых групп

Файловая группа представляет собой расширенный механизм организации объектов базы данных. По умолчанию база данных имеет всего одну файловую группу – первичную. При конфигурировании базы данных для использования нескольких файловых групп конкретные объекты (таблицы, индексы и т.п.) могут создаваться в заданных группах. Эта методика может поддерживать две основные стратегии.

* Использование нескольких файловых групп способно повысить производительность за счет разделения таблиц и индексов на разные дисковые подсистемы.
* Использование нескольких файловых групп может помочь организовать план резервирования и восстановления, помещая статические данные в одну файловую группу и активизированные данные – в другую.

#### Создание базы данных с файловыми группами

Чтобы добавить к базе данных файловую группу в Management Studio, откройте страницу параметров базы данных в Object Explorer. На странице Filegroups создайте дополнительные файловые группы. После этого на странице Files создавайте новые файлы и выбирайте для них в комбинированном списке конкретные файловые группы.

Чтобы выполнить ту же операцию программным путем, используйте параметр Filegroups. В следующем примере создается база данных NewDB c двумя файловыми группами:

CREATE DATABASE NewDB

ON

PRIMARY

(NAME = NewDB,

FILENAME = ‘d:\SQLData\NewDB.mdf’,

SIZE = 50 MB,

MAXSIZE = 5 Gb,

FILEGROWTH = 25 MB),

FILEGROUP GroupTwo

(NAME = NewDBGroup2,

FILENAME = ‘e:\SQLData\NewDBTwo.ndf’,

SIZE = 50 MB,

MAXSIZE = 5 Gb,

FILEGROWTH = 25 MB)

LOG ON

(NAME = NewDBLog,

FILENAME = ‘f:\SQLLog\NewDBLog.ndf’,

SIZE = 100 MB,

MAXSIZE = 25 Gb,

FILEGROWTH = 25 MB);

#### Модификация файловых групп

Файловые группы можно изменять точно так же, как и файлы. В Management Studio можно создавать новые файловые группы, добавлять и удалять в них файлы, а также удалять пустые группы. Однако удалить файловую группу несколько сложнее, чем сжать файл. Если в файловой группе содержатся какие-либо данные, то сжатие файла приведет к перемещению его данных в другой файл той же группы. Перед удалением файловой группы нужно удалить из нее таблицы и индексы.

С помощью редактора запросов и кода T-SQL можно создавать и удалять файловые группы. Для этого используются инструкции ALTER DATABASE ADD FILEGROUP и ALTER DATABASE REMOVE FILEGROUP. Они имеют практически такой же синтаксис, как инструкции добавления и удаления файлов.

#### Удаление базы данных

База данных может быть удалена с сервера с помощью выделения ее в Object Explorer и выбора в контекстном меню команды Delete.

Для программного удаления базы данных используйте инструкцию DROP DATABASE:

DROP DATABASE NewDB

### *Создание таблиц*

Подобно всем реляционным СУБД, SQL Server является таблично ориентированной. После создания базы данных следующим действием является наполнение ее таблицами. База данных SQL Server может содержать до 2 147 483 647 объектов, включая таблицы, так что вы можете создавать столько таблиц, сколько заблагорассудится.

#### Создание таблицы в Management Studio

Утилита Management Studio предлагает две рабочие области для создания и модификации таблиц.

* Конструктор таблиц (Table Designer) перечисляет столбцы по вертикали, размещая свойства столбцов в нижней части окна.
* Конструктор баз данных (Database Designer) – более гибкий инструмент, чем конструктор таблиц. В нем могут отображаться ограничения внешних ключей как связи с другими таблицами.

В каждом из этих инструментов графически представлена структура таблицы. Как только проектирование таблицы будет завершено, Management Studio генерирует сценарий, который применяет изменения к базе данных. Если вы модифицируете существующую таблицу, то часто данные изначально сохраняются во временной таблице, затем удаляются некоторые элементы, после чего создается новая таблица, в которую вставляются данные.

Конструктор таблиц отображает только имена столбцов и их тип данных (вместе с длиной), а также допустимость пустых значений.

Как только редактирование структуры таблицы будет выполнено, активизируется кнопка Save Change Script панели инструментов. Она приводит к отображению кода, который конструктор таблиц должен выполнить для фактического сохранения изменений. К тому же кнопка Save Change Script позволяет сохранить сценарий в файле .sql, чтобы его можно было повторить на другом сервере.

#### Работа со сценариями SQL

Если вы создаете базу данных для массового тиражирования, то преимущества реализации схемы в сценариях становиться очевидным.

* Код сценария находиться в одном месте. Работа со сценариями SQL аналогична работе с языками VB.NET и C#.
* Сценарии могут храниться в узлах решений и проектов в Solution Explorer, а также в Microsoft SourceSafe или в другой системе управления изменениями.
* Если базовый сценарий базы данных содержит весь код, необходимый для ее создания, то самая современная версия базы данных может быть установлена без необходимости запуска сценариев изменений или запуска процесса восстановления из резервной копии.
* Установка, которая освежает базу данных, в противоположность резервной копии имеет то преимущестов, что не содержит каких-либо устаревших данных.

В то же время работа со сценариями имеет и свои недостатки.

* Инструкции T-SQL могут оказаться незнакомыми, а размер сценария может быть запредельно большим.
* Если ограничения внешнего ключа внедрены в таблицу, то порядок операций при создании таблицы может оказаться критичным. Если ограничения применяются уже после создания таблиц, то этот порядок может быть несущественным, однако при этом создание внешних ключей будет в программе дистанционно отдалено от создания таблиц.
* Диаграммы базы данных Management Studio не являются частью сценария.

Для работы с объектами (в том числе и с таблицами) предназначены следующие инструкции T-SQL: CREATE, ALTER и DROP. Следующая инструкция из учебной базы данных создает таблицу ProductCategory. В ней представлено имя таблицы (вместе с именем ее владельца – dbo), за которым следуют ее столбцы. Заключительная строка указывает серверу создать таблицу в первичной файловой группе:

CREATE TABLE dbo.ProductCategory (

ProductCategoryID UNIQUEIDENTIFIER NOT NULL

ROWGUIDCOL DEFAULT (NEWID()) PRIMARY KEY NONCLUSTERED,

ProductCategoryName NVARCHAR(50) NOT NULL,

ProductCategoryDescription NVARCHAR(100) NULL

)

ON [Primary];

#### Схемы

Схема представляет собой объект, который служит исключительно для того, чтобы владеть другими объектами базы данных. Она сегментирует крупные базы данных на управляемые модули, а также может реализовывать сегментированную стратегию защиты данных.

Обычно (и по умолчанию) объекты принадлежат схеме dbo. Имя схемы является третьей частью четырехкомпонентного имени:

*Сервер.база\_данных.схема.объекта*

### *Создание первичных ключей*

Первичные и внешние ключи являются теми нитями, которые связывают отдельные таблицы в единую реляционную базы данных. Проектирование этих ключей имеет критическое влияние на производительность и полезность физической базы данных.

#### Первичные ключи

Реляционная база данных зависит от первичных ключей. Это краеугольный камень физической схемы базы данных.

На физическом уровне первичные ключи служат двум целям:

* для уникальной идентификации строк;
* как объект ссылки для внешних ключей.

В SQL Server первичные и внешние ключи реализованы как ограничения. Задачей данных ограничений являются гарантирование удовлетворения новыми данными определенных критериев или блокирование операций модификации данных.

Ограничении первичного ключа является комбинацией требований уникальности (что подразумевает и запрет пустых значений) и использования уникального индекса (кластеризованного или нет).

#### Создание первичных ключей

В программе столбец может быть назначен первичным ключом двумя способами.

* С помощью объявления ограничения первичного ключа в инструкции CREATE TABLE. Следующий пример из учебной базы данных использует этот способ для создания таблицы Guide и назначения поля GuidID первичным ключом с некластеризованным индексом:  
  CREATE TABLE dbo.Guide (  
   GuideID INT IDENTITY NOT NULL PRIMARY KEY NONCLUSTERED,  
   LastName VARCHAR(50) NOT NULL,  
   FirstName VARCHAR(50) NOT NULL,  
   Qualifications VARCHAR(2048) NULL,  
   DateOfBirth DATETIME NULL,  
   DateHire DATETIME NULL  
   )  
   ON [Primary];
* С помощью объявления первичного ключа после создания таблицы. Для этого используется инструкция ALTER TABLE. Предполагая, что первичный ключ пока не назначен для таблицы Guide, следующая инструкция позволит это сделать:  
  ALTER TABLE dbo.Guide ADD CONSTRAINT  
  PK\_Guide PRIMARY KEY NONCLUSTERED(GuideID)  
  ON [PRIMARY];

#### Естественные первичные ключи

*Естественные ключи* отражают то, как объект идентифицируется в реальном мире. Фамилии людей, номера автомобилей и паспортов, а также адреса являются примерами естественных ключей.

Естественные ключи имеют свои достоинства и недостатки.

* Естественные ключи легко распознаются людьми. По ним пользователи могут легко найти нужные данные. Недостатком является то, что люди обычно хотят придать первичному ключу некоторый смысл и вносят в него определенные символы. Люди имеют склонность изменять то, что сами создают, а модификация значений первичного ключа может вызвать проблемы. При использовании естественного первичного ключа, необходимо обязательно каскадировать обновления всех внешних ключей ссылающегося на него. Только так можно гарантировать ссылочную целостность базы данных при внесении изменений в первичный ключ.
* Естественные ключи размножают значение первичного ключа во все поколения внешних ключей. К тому же составные внешние ключи имеют склонность разрастаться, если включать в себя значения всех столбцов первичных ключей. Преимуществом является то, что можно создать объединение самой нижней в иерархии вторичной таблицы непосредственно с первичной, исключив из цепочки все промежуточные. Недостатком является громоздкость таких ключей, при этом большинство объединений должно использовать несколько столбцов.
* Естественные ключи обычно не имеют какой-либо определенной организации. Это отрицательно сказывается на производительности при вставке информации в середину отсортированных данных, что вызвано необходимостью разбиения страниц.

#### Использование столбцов идентичности в качестве суррогатных первичных ключей

Суррогатные ключи заполняются компьютером и, как правило, не имеют для людей никакого смыслового значения. В SQL Server суррогатные ключи представляют собой столбцы идентичности или глобальные уникальные идентификаторы.

До сих пор самым распространенным методом создания первичных ключей является использование столбца идентичности. Подобно столбцу автонумерации или последовательности в других базах данных, столбец идентичности при вставке в таблицу новых строк генерирует последовательность целых чисел. При желании вы можете назначить для этой последовательности изначальное число и интервал.

* Столбцы идентичности имеют два основных достоинства. Целые числа легче распознать на глаз и отредактировать, чем глобальные идентификаторы (GUID)
* Целые числа невелики и быстро обрабатываются.

Отрицательные стороны использования столбцов идентичности приведены ниже.

* Их уникальность распространяется только на одну таблицу – в разных таблицах будут использованы одни и те же значения.
* Модели со столбцами идентичности склонные использовать во всех таблицах суррогатные ключи. Это выливается в использование составных первичных ключей, созданных множеством внешних. Несмотря на то что при этом создаются маленькие и быстрые первичные ключи, для перемещения по схеме данных требуется большее число объединений.

Значения ключей идентичности генерируются ядром базы данных при вставке строки. Попытка вручную вставить значения в столбец идентичности приведет к ошибке, если, конечно, для параметра SET INSERT\_IDENTITY не установлено значение True.

В следующем примере создается таблица, которая использует столбец идентичности для своего первичного ключа:

CREATE TABLE dbo.Event (

EventID INT IDENTITY NOT NULL PRIMARY KEY NONCLUSTERED,

TourID INT NOT NULL FOREIGN KEY REFERENCES dbo.Tour,

EventCode VARCHAR(10) NOT NULL,

DateBegin DATETIME NULL,

Comment NVARCHAR(255)

)

ON [Primary];

Установка столбца (или столбцов) в качестве первичного ключа в Management Studio не сложнее выделения этого столбца и щелчка на соответствующей кнопке панели инструментов. Для создания составного первичного ключа выберите все задействованные в нем столбцы и щелкните на кнопке панели инструментов.

#### Использование уникальных идентификаторов в качестве суррогатных первичных ключей

Тип данных uniqueidentifier в SQL Server является двойником глобального уникального идентификатора (GUID), используемого в среде .NET. Это 16-байтовое шестнадцатеричное число, которое уникально для каждой таблицы, базы данных и сервера на всей планете. Хотя в пределах одного столбца уникальны и значения столбца идентичности, и значения GUID, у последнего область уникальности гораздо шире, чем у первого. Нарушая правила стилистики, можно сказать, что GUID более уникален, чем столбцы идентичности.

Идентификаторы GUID имеют ряд существенных достоинств.

* База данных, использующая GUID в качестве первичных ключей, может быть легко реплицирована без каких-либо осложнений. Репликация добавит уникальные идентификаторы во все таблицы без столбца с типом uniqueidentifier. Несмотря на то что это сделает столбцы глобально уникальными для задач репликации, код приложения будет продолжать идентифицировать строки только по целочисленным первичным ключам. Таким образом, слияние реплицированных строк в других серверах приведет к ошибке, так как будут встречаться дублирующиеся значения первичного ключа.
* Использование GUID отстраняет пользователя от смыслового значения первичного ключа.
* Использование GUID поможет избежать ошибок, вызванных некорректным объединением таблиц, но, независимо от этого, данные будут возвращаться. Причина этого лежит в том, что даже несоответствующие строки в ключевых столбцах будут использовать целочисленные значения.
* Число глобальных уникальных идентификаторов бесконечно. Таблицы, созданные на основе столбца идентичности, могут содержать только 2 147 483 648 строк. Естественно, тип данных можно установить в bigint или nemeric, однако это лишит вас всех достоинств столбца идентичности.
* Поскольку идентификаторы GUID могут генерироваться либо как значения столбца по умолчанию, либо в выражениях инструкции SELECT, их программирование существенно упрощается по сравнению с программированием столбцов идентичности. Использование GUID позволит обойти стороной все проблемы, возникающие при использовании столбца идентичности.

Недостатки использования идентификаторов GUID в основном связаны с производительностью.

* Идентификаторы GUID существенно крупнее по сравнению с целыми числами, поэтому на странице их помещается меньше. В результате извлечения одного и того же количества строк придется считывать большее количество страниц.
* Уникальные идентификаторы, генерируемые функцией NewID(), подобно естественным ключам, по своей сути случайны. Таким образом, операции вставки данных будут вызывать разбиения страниц, что негативно скажется на общей производительности. В то же время естественные ключи имеют естественное распределение, поэтому проблема разбиения страниц в естественных ключах встречается реже.

Таблица Product в ниже приведенном примере использует в качестве первичного ключа глобальный уникальный идентификатор. В следующем примере тип данных столбца ProductID устанавливается как uniqueidentifier. В нем запрещаются пустые значения, а свойство rowguidcol устанавливается в значение True, что позволяет использовать это столбец в репликациях. В качестве значения по умолчанию используется новый генерируемый идентификатор. Это поле используется в качестве первичного ключа, и для него создается некластеризованный индекс:

CREATE TABLE dbo.Product (

ProductID UNIQUEIDENTIFIER NOT NULL

ROWGUIDCOL DEFAULT (NEWSEQUNTIALID())

PRIMARY KEY CLUSTERED,

ProductCategoryID UNIQUEIDENTIFIER NOT NULL

FOREIGN KEY REFERENCES dbo.ProductCategory,

ProductCode CHAR(15) NOT NULL,

ProductName NVARCHAR(50) NOT NULL,

ProductDescription NVARCHAR(100) NULL,

ActiveDate DATETIME NOT NULL DEFAULT GETDATE(),

)

ON [Static];

Существует два основных метода генерирования уникальных идентификаторов (на самом деле все они генерируются операционной системой Windows) и множество мест, где эта генерация может выполняться.

* Функция NewID() генерирует уникальный идентификатор, используя несколько факторов, в том числе код NIC, адрес MAC, внутренний идентификатор процессора и текущий момент в часах компьютера. Последние шесть байтов занимает номер узла сетевой карты.  
  Универсальная функция NewID может использоваться в качестве значения по умолчанию для столбца, передаваться в инструкцию INSERT, а также выполняться как функция в любом выражении.
* Функция NewsequentionalID() аналогична функции NewID(), однако в отличие от последнего гарантирует, что каждое следующее значение уникального идентификатора больше всех остальных в таблице.

Функция NewsequentionalID() может использоваться только в качестве значения по умолчанию для столбца. И это имеет определенный смысл, так как генерируемое значение зависит от наибольшего идентификатора в конкретной таблице.

### *Создание внешних ключей*

Вторичная таблица, которая связана с первичной, использует внешний ключ для указания на первичный ключ первичной таблицы. Требования ссылочной целостности подразумевают, что каждый внешний ключ обязательно указывает на первичный. Поддержание ссылочной целостности жизненно важно в реляционных базах данных. База данных должна начинать и завершать каждую транзакцию в непротиворечивом состоянии. Эта целостность должна расширяться на ссылки внешних ключей.

Таблицы в SQL Server могут поддерживать до 253 ограничений внешних ключей. Внешние ключи могут ссылаться на первичные ключи, уникальные ограничения и уникальные индексы любой таблицы, за исключением, естественно, временных.

#### Декларативная ссылочная целостность

Декларативная ссылочная целостность (DRI) в SQL Server может гарантировать ссылочную целостность без необходимости написания пользователем триггеров или процедур. Она обслуживается самим ядром базы данных и практически не загружает ресурсы компьютера (в отличие от триггера).

В SQL Server декларативная ссылочная целостность поддерживается с помощью ограничений внешних ключей. Вы можете открыть форму Foreign Key Relationships, показанную на Рис. 3, чтобы создать или изменить ограничения внешних ключей. Для этого выполните следующее.

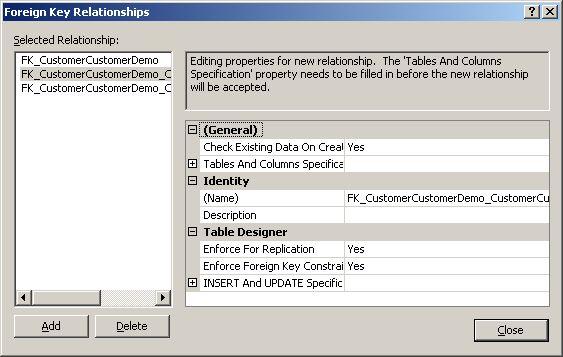
* В конструкторе баз данных выделите столбец первичного ключа и перетащите его к столбцу внешнего ключа. Это приведет к открытию диалогового окна Foreign Key Relationships.
* В конструкторе таблиц щелкните на кнопке Relationships панели инструментов или выберите в меню команду Table -> Designer -> Relationships. В качестве альтернативы можете выбрать в конструкторе баз данных вторую таблицу (с внешним ключом), а затем щелкнуть на кнопке Relationships панели инструментов или выбрать одноименный пункт из контекстного меню этой таблицы.

В окне Foreign Key Relationships существует несколько параметров, позволяющих определить режим работы внешнего ключа.

* Enforce for Replications (поддерживать репликации).
* Enforce Foreign Key Constraint (поддерживать ограничения внешнего ключа).
* Delete Rule (каскадное удаление).
* Update Rule (обновить правило).

В коде T-SQL вы можете объявить ограничения внешнего ключа либо при создании таблиц, либо после этого. После определения столбца фраза FOREIGN KEY REFERENCE, за которой следует имя таблицы (и при необходимости – столбца), позволяет создать внешний ключ, например:

ForeignKeyColumn FOREIGN KEY REFERENCES PrimaryTable(PKID)



**Рис. 5. Форма Foreign Key Relationships утилиты Management Studio используется для создания и модификации декларативной ссылочной целостности**

В следующем примере создается связующая таблица tour\_mm\_guide, разрешающая отношение «многие ко многим». Эта таблица имеет два внешних ключа: по одному для таблиц Tour и Guide. Внешний ключ TourID явно ссылается на столбец первичного ключа, а внешний ключ GuidID указывает на таблицу и использует первичный ключ по умолчанию:

CREATE TABLE dbo.Tour\_mm\_Guide (

TourGuideID INT

IDENTITY

NOT NULL

PRIMARY KEY NONCLUSTERED,

TourID INT

NOT NULL

FOREIGN KEY REFERENCES dbo.Tour(TourID)

ON DELETE CASCADE,

GuideID INT

NOT NULL

FOREIGN KEY REFERENCES dbo.Guide

ON DELETE CASCADE,

QualDate DATETIME NOT NULL,

RevokeDate DATETIME NULL

)

ON [Primary];

Если таблица уже существует, то вы можете создать ограничение первичного ключа, используя инструкцию ALTER TABLE ADD CONSTRAINT:

ALTER TABLE SecondaryTableName

ADD CONSTRAINT ConstraintName

FOREIGN KEY (ForeignKeyColumns)

REFERENCES dbo.PrimaryTable (PrimaryKeyColumnName);

#### Каскадные удаления и обновления

Сложности, создаваемые ссылочной целостностью, сводятся к недопустимости удаления или модификации строки первичной таблицы до тех пор, пока существуют строки вторичной таблицы, ссылающиеся на нее (т.е. на строку). Если последние остаются после удаления строки первичной таблицы, то нарушаются требования ссылочной целостности.

Решение этой проблемы сводится к модификации строк вторичной таблицы как части транзакции, изменяющей строки первичной таблицы. Декларативная ссылочная целостность может выполнить эту работу вместо вас. Для строк вторичной таблицы возможны четыре варианта действий, выбираемых в параметрах Delete Rule и Update Rule формы Foreign Key Relationships. Параметр Update Rule имеет смысл только для естественных первичных ключей.

* **No Action**. Строки вторичной таблицы не изменяются. Их присутствие будет предохранять строки первичной таблицы от удаления и модификации.  
  Используйте этот вариант, когда вторичные строки сопровождают значения первичных строк. Вы не сможете изменить или удалить первичные строки, если существуют связанные с ними вторичные. Например, если для какого-то клиента выписан счет-фактура, вы не сможете его удалить.
* **Cascade**. Операции удаления и модификации, выполняемые на первичных строках выполняются также и на вторичных.  
  Используйте этот вариант, когда вторичные данные теряют смысл в отсутствие первичных. Например, если удаляется заказ с номером 123, все его строки будут также удаляться. Если номер заказа изменяется с 123 на 456, то его строки также должны изменить номер заказа на это значение (предполагается, что используется естественный первичный ключ).
* **Set Null**. Этот вариант оставляет вторичные строки нетронутыми, однако изменяет значения вторичного ключа на пустое (разумеется, предполагается допустимость наличия пустых значений).  
  Используйте этот вариант, если хотите разрешить удаление первичных строк без влияния на существование вторичных. Например, если удаляется запись о школьном классе, строки с данными об учениках не должны удаляться, поскольку их существование в базе данных самодостаточно.
* **Set Default**. Первичные строки могут удаляться и изменяться, при этом значения внешнего ключа изменяются на значения по умолчанию, заданные для данного столбца.  
  Этот вариант аналогичен варианту Set Null, за исключение того, что на этот раз внешнему ключу присваивается определенное значение. В схемах, использующих суррогатные пустые значения (например, пустые строки), установка значений столбца по умолчанию и определение правила удаления как Set Default приведет к занесению во внешний ключ пустой строки при удалении строки первичного ключа.

В коде T-SQL добавление параметра ON DELETE CASCADE в ограничение внешнего ключа форсирует каскадные операции. В следующем примере таблица строк заказа OrderDetail использует параметр каскадного удаления в ограничении внешнего ключа OrderID:

CREATE TABLE dbo.OrderDetail (

OrderDetailID UNIQUEIDENTIFIER

NOT NULL

ROWGUIDCOL

DEFAULT (NEWID())

PRIMARY KEY NONCLUSTERED,

OrderID UNIQUEIDENTIFIER

NOT NULL

FOREIGN KEY REFERENCES dbo.[Order]

ON DELETE CASCADE,

ProductID UNIQUEIDENTIFIER

NULL

FOREIGN KEY REFERENCES dbo.Product,

### *Создание пользовательских столбцов данных*

Эти столбцы хранят данные пользователя. Формально их можно разделить на две категории: используемые для идентификации людей, мест, событий или действий и используемые для дополнительного описания людей, мест, событий и действий.

Таблицы SQL Server могут содержать до 1024 столбцов; в то же время считается, что в таблице хорошо сформированной реляционной базы данных их число не должно превышать двадцати пяти.

Столбцы данных создаются в процессе определения таблицы и перечисляются в инструкции CREATE TABLE. Столбцы перечисляются в круглых скобках, при этом указывается имя столбца, используемый в нем тип данных, а также дополнительные атрибуты, ограничения, допустимость пустых значений и значения по умолчанию. Общий синтаксис следующий:

CREATE TABLE имя\_таблицы (

имя\_столбца тип\_данных атрибуты,

имя\_столбца тип\_данных атрибуты

);

Столбцы данных могут добавляться и в существующие таблицы с помощью инструкции ALTER TABLE ADD COLUMN:

ALTER TABLE имя\_таблицы

ADD имя\_столбца тип\_данных атрибуты;

Существующие столбцы могут быть изменены с помощью инструкции ALTER TABLE ALTER COLUMN:

ALTER TABLE имя\_таблицы

ALTER COLUMN имя\_столбца

новый\_тип\_данных атрибуты;

### *Типы данных столбцов*

Тип данных столбца служит для двух целей.

* Он обеспечивает первый уровень целостности данных. Символьные данные не могут быть занесены в столбцы типов даты-времени и числового типа. Тип данных играет важную роль как инструмент проверки допустимости данных, и его лучше не обходить вниманием.
* Он определяет объем дискового пространства, выделяемого для столбца.

#### Символьные типы данных

SQL Server поддерживает несколько символьных типов данных.

Типы данных, использующие таблицу Unicode, особенно полезны для хранения текста, написанного на нескольких языках. Однако за это приходится расплачиваться удвоением занимаемого пространства.

**Табл. 11. Символьные типы данных**

|  |  |  |
| --- | --- | --- |
| **Тип данных** | **Описание** | **Размер в байтах** |
| Char (n) | Символьные данные фиксированной длины. Могут содержать до 8000 символов и использовать принятый по умолчанию порядок и набор символов | Заданная длина, умноженная на 1 байт |
| Nchar(n) | Символьные данные фиксированной длины в таблице Unicode | Заданная длина, умноженная на 2 байта |
| Varchar(n) | Символьные данные переменной длины. Могут содержать до 8000 символов и использовать принятый по умолчанию порядок и набор символов | По 1 байту на символ |
| varchar(max) | Символьные данные переменной длины. Могут содержать до 2 Гбайт информации и использовать принятый по умолчанию порядок и набор символов | По 1 байту на символ |
| nvarchar(n) | Символьные данные переменной длины, хранящие до 8000 символов при использовании порядка, принятого по умолчанию | По 2 байта на символ |
| nvarchar(max) | Символьные данные переменной длины, хранящие до 2 Гбайт. Используется таблица Unicode и порядок, принятый по умолчанию | По 2 байта на символ |
| text | Символьные данные переменной длины, содержащие до 2 147 483 647 | По 1 байту на символ |
| ntext | Символьные данные в таблице Unicode, содержащие до 1 073 741 823 символов | По 2 байта на символ |
| sysname | Пользовательский тип данных, используемый для имен таблиц и столбцов, - эквивалент типа nvarchar(128) | По 2 байта на символ |

#### Числовые типы данных

SQL Server поддерживает несколько числовых типов данных

**Табл. 12. Числовые типы данных**

|  |  |  |
| --- | --- | --- |
| **Тип данных** | **Описание** | **Размер в байтах** |
| bit | 1 или 0 | 1 бит |
| tinyint | Целые числа от 0 до 255 | 1 байт |
| smallint | Целые числа от -32768 до 32767 | 2 байта |
| int | Целые числа от -2 147 483 648 до 2 147 483 647 | 4 байта |
| bigint | Целые чилса от -263 до 263 | 8 байтов |
| decimal или nemeric | Числа с фиксированной точностью до 1038+1 | Варьируется в зависимости от длины |
| money | Числа от -263 до 263 с точностью до одной десятитысячной | 8 байтов |
| smallmoney | Числа от -2 14748,3648 до 214748,3647 | 4 байта |
| float | Числа с плавающей запятой от -1,79E+308 до 1,79E+308 | 4 байта или 8 байтов в зависимости от точности |
| real | Числа с плавающей запятой с точностью до 24 знаков | 4 байта |

#### Типы данных даты-времени

SQL Server хранит в одном столбце дату и время и использует для этого типы данных datetime и smalldatetime. Основным различием между этими двумя типами является точность и учет столетия. Если в столбце должна храниться только дата и при этом не должен учитываться период до XX века, то вполне подойдет тип smalldatetime. Если требуется дополнительно хранить значения времени, точность типа smalldatetime может не хватить.

**Табл. 13. Типы данных даты-времени**

|  |  |  |
| --- | --- | --- |
| **Тип данных** | **Описание** | **Размер в байтах** |
| datetime | Значения даты и времени от 1 января 1753 года до 31 декабря 9999 года с точностью до трех миллисекунд | 8 байтов |
| smalldatetime | Значение даты и времени с 1 января 1900 года до 6 июня 2079 года с точностью до одной минуты | 4 байта |

#### Прочие типы данных

Прочие типы данных перечислены и описаны в Табл. 5. Прочие типы данных. Они способны покрыть потребности двоичных объектов и вариантных данных.

**Табл. 14. Прочие типы данных**

|  |  |  |
| --- | --- | --- |
| **Тип данных** | **Описание** | **Размер в байтах** |
| timestamp | Уникальное в пределах базы данных случайное значение, генерируемое при каждом обновлении | 8 байтов |
| uniqueidentifier | Генерируемое системой 16-байтовое значение | 16 байтов |
| binary(n) | Данные фиксированной длины вплоть до 8000 байтов | Определяется длиной |
| binary(max) | Данные фиксированной длины вплоть до 8000 байтов | Определяется длиной |
| varbinary | Двоичные данные переменной длины вплоть до 8000 байтов | Число используемых байтов |
| image | Двоичные данные переменной длины вплоть до 2 147 483 647 байтов | Число используемых байтов |
| sql\_variant | Может хранить любые типы данных длиной до 2 147 483 647 байтов |  |

Вычисляемые столбцы

Вычисляемые столбцы имеют в таблицах особую ценность, поскольку содержат результат предопределенного выражения так же, как и представления (т.е. сохраненные инструкции SELECT), однако без излишней нагрузки на систему. Такие столбцы не хранят реальных данных – они их вычисляют в момент выполнения запроса.

Вычисляемые столбцы также положительно влияют на целостность данных. Они вычисляют значения на уровне таблицы, не доверяя эту операцию каждому конкретному пользователю. По ним могут даже создаваться индексы.

Синтаксис вычисляемых столбцов противоположный присвоению псевдонима:

*Имя\_столбца AS выражение*

Таблица OrderDetails ниже приведенного примера содержит вычисляемый столбец для расширенной цены, что и продемонстрировано далее в ее определении:

CREATE TABLE dbo.OrderDetails (

…

Quantity NUMERIC(7, 2) NOT NULL,

UnitPrice MONEY NOT NULL,

ExtendedPrice AS Quantity \* UnitPrice Persisted,

…

)

ON [Primary];

Go

### *Ограничения и значения столбцов по умолчанию*

База данных всегда настолько хороша, насколько хороши ее данные. Ограничения представляют собой высокоскоростную проверку допустимости значений или правил бизнес-логики, выполняемую на уровне ядра базы данных. Кроме проверки типа данных, SQL Server содержит пять типов ограничений.

* **Ограничения первичного ключа**. Гарантирует уникальность первичного ключа и отсутствия в нем пустых значений.
* **Ограничения внешнего ключа**. Гарантирует указание значения на допустимый ключ.
* **Допустимость пустых значений**. Проверяют наличие в столбце пустых значений, если они не допустимы.
* **Ограничения проверки**. Эти булевы ограничения задаются пользователем.
* **Ограничения на уникальность**. Гарантируют уникальность значений.

SQL Server также предлагает использование параметров столбцов.

* **Значение по умолчанию**. Если инструкция INSERT не вставляет в столбец никакого значения, в него подставляется значение по умолчанию.

#### Допустимость пустых значений

Пустое значение можно расценивать как неизвестное. Как правило, пустые значения появляются при неполном вводе пользователем данных в строку.

Допустимы ли в столбце пустые значения, определяется с помощью атрибута столбца NULL или NOT NULL.

По умолчанию в новых столбцах SQL Server предполагает недопустимость пустых значений.

#### Ограничение на уникальность

Ограничение на уникальность аналогично уникальному индексу или ограничению первичного ключа. Оно гарантирует то, что в разных строках столбца не будет содержаться одинаковые значения. Этот параметр чаще всего используется, когда в таблице нужно обеспечить отсутствие дублирующихся строк (например, в списке сотрудников уникальным должен быть идентификационный код).

В утилите Management Studio ограничение на уникальность устанавливается во вкладке Index диалогового окна свойств таблицы. Этот процесс идентичен установке индекса, но с одним отличием: в этом случае вместо индекса выбирается ограничение.

В программном коде ограничение на уникальность можно установить, установив в определении столбца параметр UNIQUE. Приведем пример:

CREATE TABLE Employee (

EmployeeID INT PRIMARY KEY NONCLUSTERED,

EmployeeNumber CHAR(8) UNIQUE,

LastName NVARCHAR(35),

FirstName NVARCHAR(35)

);

Чтобы добавить ограничение на уникальность в уже существующую таблицу, можно использовать инструкцию ALTER TABLE:

ALTER TABLE Employee

ADD CONSTRAINT EmpNumUnique

UNIQUE (EmployeeNumber);

##### Ограничения проверки

Ограничения проверки являются проверкой целостности данных на уровне строк. Как правило, это небольшая формула, которая возвращает булево значение true или false. Ограничения проверки имеют доступ ко всем данным текущей строки – они не могут обратиться к другим строкам или выполнить поиск. В ограничения проверки могут быть включены скалярные функции.

Ограничения проверки полезны для поддержания общих правил допустимости данных, а также простых правил бизнес-логики. В качестве примеров можно привести проверку превышения датой увольнения даты примем на работу (или даты рождения плюс 18 лет).

В следующем примере ограничение обеспечивает превышение табельным номером сотрудника (поле EmployeeNumber) значения 1:

CREATE TABLE Employee (

EmployeeID INT PRIMARY KEY NONCLUSTERED,

EmployeeNumber CHAR(8) CHECK (EmployeeNumber > ‘1’),

LastName NVARCHAR(35),

FirstName NVARCHAR(35)

);

#### Значения по умолчанию

Это значение SQL Server вставляет в таблицу, если оно не было определено в инструкции INSERT в явном виде. Значения по умолчанию приобретают особое значение, когда в столбце недопустимы пустые значения. В этом случае, если не вставить в такой столбец значение, а значения по умолчанию в нем не определены, вся операция вставки будет отвергнута.

Значения по умолчанию может быть одним из следующих.

* Допустимое статическое числовое или символьное значение, такое как 123 или local.
* Скалярная системная функция, такая как GetDate() или NewID().
* Скалярная функция, определенная пользователем.
* Пустое значение.

Тип значения по умолчанию должен быть совместим с типом данных столбца.

Если таблица создается в Management Studio, то значение по умолчанию определяется как одно из свойств столбца.

В программном коде значение по умолчанию добавляется как один из параметров определения столбца при создании таблицы или уже впоследствии, с помощью инструкции ALTER TABLE CREATE CONSTRAINT.

Следующий пример приведен для определения таблицы Product. Значением столбца по умолчанию ActiveDate является текущая дата:

CREATE TABLE dbo.Product (

…

ActiveDate DATETIME NOT NULL DEFAULT GETDATE(),

…

);

Значения по умолчанию можно устанавливать и после создания таблицы.

ALTER TABLE Product

ADD CONSTRAIN ActiveDefault

DEFAULT GetDate() FOR ActiveDate;

#### Правила, определяемые пользователем

Правило аналогично ограничению проверки, за исключение того, что создается независимо и только затем применяется к столбцу. Правило состоит только из имени и булева выражения. Булево выражение может ссылаться на данные с помощью символа @, за которым следует им столбца.

В следующем примере продемонстрировано создание правила, которое проверяет дни рождения и гарантирует, что все они в прошлом:

-- Определяемое пользователем правило

CREATE RULE BirthdateRule AS @Birthdate <= Getdate();

Чтобы применить это правило к столбцу таблицы или пользовательскому типу данных, используется хранимая процедура sp\_bindrule. Первым аргументом этой процедуры является имя правила, а вторым – объект, к которому оно применяется. В следующем примере вышеописанное правило BirthdayRule применяется к столбцу BirthDate таблицы Person:

EXEC sp\_bindrule

@rulename = ‘BirthdateRule’,

@objname = ‘Person.Birthdate’;

В утилите Management Studio правила создаются и применяются в узле Rules каждой из баз данных.

### *Манипулирование данными с помощью инструкции SELECT*

#### Синтаксическая организация инструкции запроса

В своей базовой форме инструкция SELECT сообщает серверу, какие данные следует извлечь, в частности, какие столбцы и строки из каких таблиц получить и как сортировать данные.

Ниже приведен упрощенный синтаксис инструкции SELECT.

SELECT \*, столбцы или выражения

[FROM таблица]

[JOIN таблица

ON условие]

[WHERE условия]

[GROUP BY столбцы]

[HAVING условия]

[ORDER BY столбцы];

Инструкция SELECT начинается со списка столбцов или выражений. Как минимум обязательно наличие хотя бы одного выражении, все остальное – необязательно. Простейшая из возможных инструкций SELECT выглядит следующим образом:

SELECT 1;

Предложение FROM в инструкции SELECT собирает все источники данных в единый набор, над которым будет работать остальная часть инструкции. В предложении FROM может участвовать множество таблиц, которые ссылаются друг на друга с помощью одного из нескольких типов объединений.

Предложение WHERE фильтрует строки набора данных, собранного предложением FROM, на основе некоторых условий.

Агрегатные функции выполняют в наборе данных итоговые подсчеты. Предложение GROUP BY может группировать большие множества в несколько небольших подмножеств на основе значений столбцов, упомянутых в этом предложении. Затем агрегатные функции применяются к этим небольшим подмножествам данных, после чего результаты агрегатных функций фильтруются с помощью предложения HAVING.

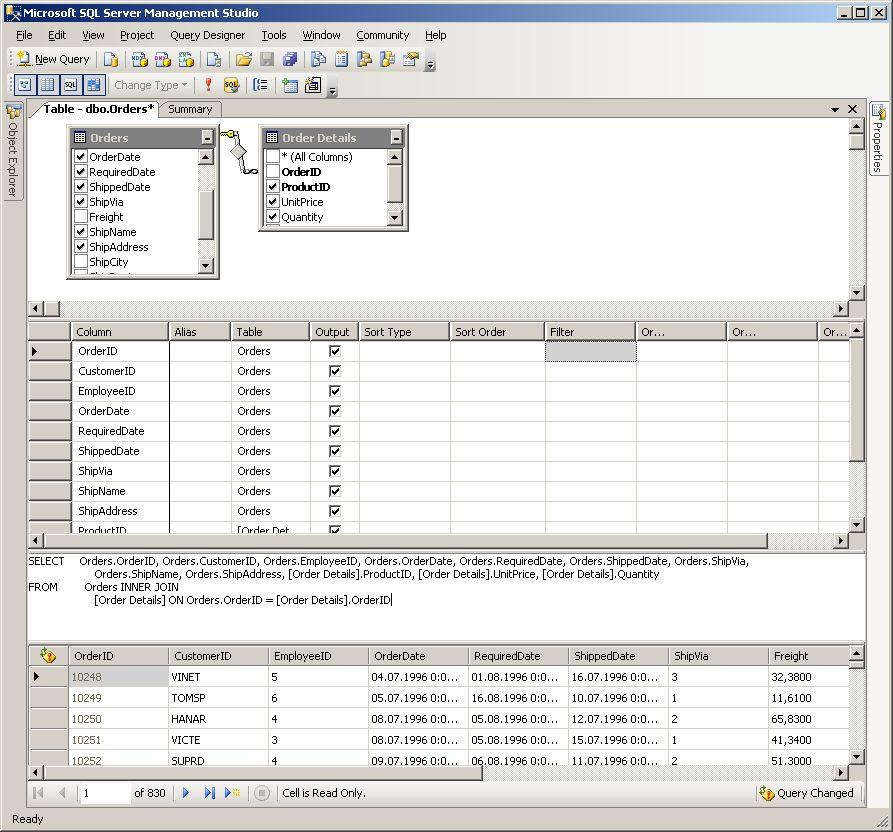
Наконец, предложение ORDER BY определяет порядок сортировки результирующего набора данных.

#### Графическое представление инструкции запроса

Утилита SQL Server Management Studio содержит два основных инструмента форматирования и отправки запросов: конструктор запросов (Query Designer) и редактор запросов (Query Editor). Конструктор запросов предлагает графический метод создания запроса. В то же время редактор запросов является идеальным средством разового извлечения данных, так как не содержит графики, указывающий пользователю, как именно он должен создавать запрос. В редакторе пользователь работает с кодом SQL настолько близко, насколько это возможно.

С точки зрения SQL Server не имеет значения, откуда именно исходит запрос. Каждая инструкция оценивается и обрабатывается как одна инструкция SQL.

Когда данные отбираются с помощью конструктора запросов, инструкции SQL можно вручную вводить и редактировать на третьей панели (Рис. 4). Нижняя панель отображает результаты выполнения запроса в табличном или текстовом режиме; также на ней отображаются различные сообщения. В окне Object Browser отображается дерево всех объектов SQL Server, равно как и шаблоны для создания новых объектов с помощью кода.



**Рис. 6. Создание запроса отбора**

#### Предложение FROM для выбора источников данных

Первым компонентом типичной инструкции SELECT является предложение FROM. В простых инструкциях отбор предложение FROM содержит всего одну таблицу. Однако это предложение может содержать и множество связанных между собой таблиц, подзапросы в качестве временных таблиц, а также представления. Максимальное количество таблиц, доступных одной инструкции SELECT, составляет 256.

Предложение FROM закладывает фундамент всех последующих операций инструкции SELECT. Для того чтобы столбец таблицы содержался в результирующем наборе данных, или был доступен условиям предложения WHERE, или стал основой сортировки в предложении ORDER BY, эта таблица должна быть упомянута в предложении FROM.

#### Возможные источники данных

Язык SQL является чрезвычайно гибким и может принимать данные из множества различных типов источников, упомянутых в предложении FROM.

* Таблицы SQL Server.
* Подзапросы, выступающие в роли временных таблиц, также называемые подвыборками и оперативными представлениями.
* Общие табличные представления (CTE), впервые введенные в SQL Server 2005, добавляют новые функции и форматирование в традиционные подзапросы.
* Представления, или хранимые инструкции SELECT, доступны предложению FROM так, будто они являются обычными таблицами.
* Определенные пользователем функции, возвращающие таблицы.
* Распределенные источники данных, перемещение из других баз и приложений (например, Oracle, Excel или Access) с помощью функции openquery() и т.д.
* Работа с источниками данных XML выполняется с помощью запросов XQuery.

#### Именованные диапазоны

Любой таблице в предложении FROM может быть присвоен именованный диапазон, или псевдоним. Как только у таблицы появляется псевдоним, по нему к ней можно обращаться в других предложениях инструкции SELECT. Ключевое слово AS является необязательным. В следующем примере доступ осуществляется к таблице Guide, но обращение к ней выполняется по псевдониму G:

SELECT G.LastName, G.FirstName

FROM Guide AS G

#### Имя таблицы

Если имя объекта базы данных, такого как таблица или столбец, конфликтует с каким либо ключевым словом SQL, вы можете указать серверу, что это именно имя объекта, заключив его в квадратные скобки.

#### Четырехкомпонентные имена таблиц

Полное и правильное имя таблицы складывается из четырех составных частей:

Сервер.База\_данных.Схема.Таблица

Если таблица находится в текущей базе данных, то имена сервера и базы не обязательны. Хотя это и не обязательно, но считается хорошим тоном указывать явно имя схемы. Обычно схемой является dbo – это наследие предыдущих версий SQL Server, где все объекты принадлежали владельцам, а имя dbo представляла владельца базы данных.

#### Условия WHERE

Условия WHERE фильтруют набор данных, сформированный предложением FROM, и отбирают из него только те строки, которые войдут в результирующий набор данных. Условия могут ссылаться на данные в таблицах, выражения, встроенные скалярные функции SQL и пользовательские функции. В условиях WHERE могут также использоваться операторы сравнения и символы макроподстановки, перечисленные в Табл. 6.

**Табл. 15. Стандартные операторы сравнения**

|  |  |  |
| --- | --- | --- |
| Описание | Оператор | Пример |
| Равно | = | Quantity=12 |
| Больше | > | Quantity>12 |
| Больше или равно | >= | Quantity>=12 |
| Меньше | < | Quantity<12 |
| Меньше или равно | <= | Quantity<=12 |
| Не равно | <>, != | Quantity<>12, Quantity!=12 |
| Не меньше | !< | Quantity !< 12 |
| Не больше | !> | Quantity !> 12 |

#### Использование условия BETWEEN

Условие BETWEEN проверяет значение не его принадлежность некоторому диапазону. В данном случае диапазон включает граничные значения. Например, условие between 1 and 10 будет справедливо для чисел 1 и 10. При использовании условия between первая граница диапазона должна быть меньше второй. Чаще всего условие between используют с датами.

Пример:

SELECT EventCode, DateBegin

FROM dbo.Event

WHERE DateBegin BETWEEN ‘07/01/01’ AND ‘07/31/01’;

#### Использование условия IN

Условие поиска in аналогично оператору сравнения equals, однако в данном случае ищется соответствие данным из списка. Если значение содержится в списке, то результатом выражения будет true.

Пример:

SELECT BaseCampname

FROM dbo.BaseCamp

WHERE Region IN(‘NC’, ‘WV’);

Оператор in можно комбинировать с оператором not для исключения соответствующих строк.

Например:

SELECT BaseCampname

FROM dbo.BaseCamp

WHERE Region NOT IN(‘NC’, ‘WV’);

#### Использование условия LIKE

Условие like использует символы макроподстановки для поиска строк, соответствующих заданному шаблону. В 16 представлены символы макроподстановки SQL.

В следующем примере условие like используется для поиска всех товаров, название которых начинаются с ‘Air’:

SELECT ProductName

FROM dbo.Product

WHERE ProductName LIKE ‘Air%’;

**Табл. 16. Символы макроподстановки SQL**

|  |  |  |
| --- | --- | --- |
| Описание | Символ SQL | Пример |
| Несколько символов | % | ‘Able’ LIKE ‘A%’ |
| Один символ | \_ | ‘Able’ LIKE ‘Abl\_’ |
| Вхождение в диапазон символов | [] | ‘a’ LIKE ‘[a-g]’ |
| Невхождение в диапазон символов | [^] | ‘a’ LIKE ‘^[w-z]’  ‘a’ LIKE ‘[^wxyz]’ |

#### Множественное условие WHERE

В предложении WHERE можно комбинировать множество условий с помощью логических булевых операторов and, or и not. Наивысший приоритет имеет оператор and, далее следует or и, наконец, not. Рассмотрим пример:

SELECT ProductCode, ProductName

FROM dbo.Product

WHERE

ProductName LIKE ‘Air%’

OR

ProductCode BETWEEN ‘1018’ AND ‘1020’

AND

ProductName LIKE ‘%G%’;

Если в выражении использовать скобки, то результат запроса радикально изменится:

SELECT ProductCode, ProductName

FROM dbo.Product

WHERE

(ProductName LIKE ‘Air%’

OR

ProductCode BETWEEN ‘1018’ AND ‘1020’)

AND

ProductName LIKE ‘%G%’;

#### Упорядочение результирующего набора данных

Данные в таблице SQL имеют форму неупорядоченного списка. Основной задачей первичного ключа является уникальная идентификация строк, а совсем не их упорядочение. Некоторые реализации СУБД могут представлять таблицы в порядке, поддерживаемом первичным ключом. Однако лучше все же не надеяться на такое поведение. Если явно не задать предложение ORDER BY, то порядок строк в результирующем наборе данных может оставаться неопределенным.

Если предложение ORDER BY отсутствует, SQL Server вернет строки в том порядке, в котором они извлекались из таблицы. Если исходная таблицы имела кластеризованный индекс, то порядок результирующего набора данных будет соответствовать ему. Некоторые логические операции сортируют данные для своей поддержки. Например, некоторые объединения сортируют данные так, чтобы объединение было легче выполнить.

SQL позволяет выполнить сортировку по множеству столбцов, и ими не обязательно должны быть столбцы, извлекаемые инструкцией SELECT. Это предоставляет достаточную гибкость в определении столбцов сортировки.

#### Определение порядка сортировки с помощью имен столбцов

Простейшим способом сортировки результирующего набора данных является явное указание порядка столбцов, по которым выполняется упорядочение:

SELECT FirstName, LastName

FROM dbo.Customer

ORDER BY LastName, FirstName;

#### Определение порядка сортировки с помощью выражений

При сортировке по выражению оно должно быть полностью повторено в предложении ORDER BY. Это не приводит к снижению производительности, поскольку оптимизатор запросов достаточно умен, чтобы не повторять одно и то же вычисление дважды:

SELECT LastName + ', ' + FirstName

FROM dbo.Customer

ORDER BY LastName + ‘, ’ + FirstName;

#### Определение порядка сортировки с помощью псевдонимов столбцов

В качестве альтернативы для явного задания столбцов в предложении ORDER BY можно использовать их псевдонимы. Этот метод предпочтительнее, так как значительно облегчает чтение программы. Обратите внимание на то, что в следующем примере сортировка выполняется по убыванию (предикат DESC), а не по возрастанию, как принято по умолчанию:

SELECT LastName + ‘, ’ + FirstName as FullName

FROM dbo.Customer

ORDER BY FullName DESC

#### SELECT DISTINCT

Первым предикатом, используемым в сочетании с ключевым словом SELECT, является DISTINCT. Он исключает дублирование строк в результирующем наборе данных запроса. Эти дублирования оцениваются на уровне столбцов результирующего набора данных, а не исходных таблиц. Противоположную функцию выполняет предикат ALL; так как он используется по умолчанию, в запросах его обычно игнорируют.

В следующем примере демонстрируется использование предикатов DISTINCT и ALL. Так как инструкция SELECT возвращает только столбец tourname (названия тура), это отличный пример дублирования строк, которое можно будет устранить с помощью предиката DISTINCT:

SELECT ALL TourName

FROM Event

JOIN Tour

ON Event.TourID = Tour.TourID

А теперь тот же запрос с предикатом DISTINCT:

SELECT DISTINCT TourName

FROM Event

JOIN Tour

ON Event.TourID = Tour.TourID

#### Ранжирование

По определению инструкция SELECT работает с наборами данных, однако иногда пользователя интересуют только первые несколько строк этого набора. Для таких ситуаций в SQL Server предусмотрено несколько способов фильтрации результатов и выявления экстремальных строк.

##### TOP

Как уже говорилось ранее, SQL Server по умолчанию возвращает в инструкции SELECT все строки результирующего набора данных. Необязательный предикат TOP указывает серверу возвращать только определенное количество строк (в абсолютном или процентном выражении), основываясь на заданном параметре.

Предикат TOP работает рука об руку с предложением ORDER BY, так как именно оно определяет, какие строки будут первыми в результирующем наборе данных. Если же в инструкции SELECT отсутствует предложение ORDER BY, то предикат TOP все равно отработает, возвращая заданное количество строк неупорядоченного набора данных.

##### Параметр WITH TIES

Параметр WITH TIES исключительно важен для предиката TOP. Он позволяет дополнить строку, занявшую последнее место в ранжировке, дополнительными строками, имеющими такое же значение в столбцах, упомянутых в предложении ORDER BY, но не попадающими в количество, заданное в предикате TOP.

#### Оператор CASE

Команда CASE в SQL Server – гибкое и удобное средство создания динамических выражений. Она используется не для программного переключения управления, а для логического определения значения выражения на основе заданного условия, подобно функции if() в других языках программирования.

Подобно всем другим выражениям, CASE не может автоматически задать имя столбца. Таким образом, как правило, каждое выражение CASE имеет псевдоним.

Так как оператор CASE возвращает, его можно использовать в любом месте любой инструкции SQL DML (SELECT, INSERT, UPDATE и DELETE), где может использоваться обычное выражение (например, выражения для столбцов, а также предложения JOIN, WHERE, HAVING и ORDER BY).

Оператор CASE имеет две формы: простую и булеву.

##### Простая форма оператора CASE

В простой форме оператора CASE вначале представлено само значение, после чего перечисляются все тестовые значения. Однако эта форма ограничена тем, что можно использовать только сравнения на предмет равенства. Данный оператор CASE последовательно проверяет все условия WHEN и возвращает значения THEN из первой строки, в которой выполнится условие WHEN.

В следующем примере, один тип (поле customertype) назначается по умолчанию для всех новых клиентов, при этом в столбец isdefault заносится true. Оператор CASE сравнивает значения в столбце isdefault со всеми возможными значениями бита, после чего возвращает символьную строку 'default type' или 'possible' в зависимости от текущего значения:

SELECT CustomerTypeName,

CASE [IsDefault]

WHEN 1 THEN ‘default type’

WHEN 0 THEN ‘possible’

ELSE ‘-’

END as AssignStatus

FROM CustomerType

Оператор CASE завершается ключевым словом END и псевдонимом. В данном примере оператор CASE тестировал значения столбца isdefault, но формировал столбец AssignStatus в результирующем наборе данных инструкции SELECT.

##### Булева форма оператора CASE

Булева форма оператора CASE более гибкая, чем простая. В ней каждый оператор WHEN имеет свое условие. Таким образом, имеется возможность не только использовать условные операторы, отличные от равенства, но и ссылаться на разные столбцы.

SELECT

CASE

WHEN 1<0 THEN ‘Реальность призрачна.’

WHEN GetDate() =’11/30/2005’

THEN ‘Дейвид получил водительские права.’

WHEN 1>0 THEN 'Жизнь продолжается.'

END AS RealityCheck

#### Работа с пустыми значениями

Реляционная модель базы данных представляет отсутствие данных с помощью специального значения NULL. В переводе на обычный язык его можно перевести так: «Значение не известно». На практике такие ситуации возникают, когда данные еще не ведены полностью или когда данный столбец не применим к конкретной строке. Фактически NULL представляет собой неопределенное или пустое значение.

Поскольку значение NULL не известно, то и результат любой операции, включающей в себя NULL, не может быть известен. Если величина банковского счета не известна, а он включен в общий список состояния, то и общая величина состояния не известна.

SELECT 1 + NULL

Результатом будет NULL.

##### Проверка на пустые значения

Так как значение NULL не известно, одно значение NULL не может быть равно другому значению NULL. Возвращаясь к примеру с банковскими счетами, предположим, что величина счета 123 не известна и величина счета 234 также не известна. Так каким же образом можно утверждать, что состояния этих счетов равны? Поскольку оператор равенства (=) не применим к пустым значения, в языке SQL введен специальный оператор IS, который используется для тестирования на равенство специальным значениям. Например:

WHERE выражение IS NULL

Условие IS NULL используется для тестирования на пустые значения.

IF NULL IS NULL

SELECT ‘IS’

ELSE

SELECT ‘IS NOT’

Результат данного выражения – ‘IS’.

Оператор IS можно комбинировать с оператором NOT для тестирования на наличие значения. Например, для отбора только тех клиентов, у которых есть псевдонимы, можно использовать условие Nickname IS NOT NULL:

SELECT FirstName, LastName, Nickname

FROM dbo.Customer

WHERE Nickname IS NOT NULL

ORDER BY LastName, FirstName

##### Обработка пустых значений

Когда данные отбираются для отчетов, конечных пользователей или приложений, пустые значения редко приветствуются. Часто пустое значение нужно преобразовать в некоторое допустимое, чтобы данные можно было понять или чтобы выражение имело результат.

Пустые значения требуют специальной обработки при использовании в выражениях, и язык SQL содержит ряд функций, специально предназначенных для работы с пустыми значениями. Функции Isnull() и coalesce() преобразует пустые значения в пригодные для использования, а функция nullif() создает пустое значение, если выполняется определенное условие.

##### Использование функции IsNull()

Наиболее часто используемой функцией, предназначенной для работы с пустыми значениями, является IsNull(), которая на самом деле отличается от условия IS NULL. Эта функция в качестве аргумента принимает одно выражение или столбец, а также подстановочное значение. Если первый аргумент является допустимым значением (т.е. не пустым) , эта функция возвращает его. Однако если первый аргумент представляет собой пустой значение, то возвращается значение второго аргумента. Общий синтаксис функции следующий:

IsNull (исходное\_выражение, замещающее\_значение)

##### Функция Coalesce()

Эта функция используется довольно редко, возможно, потому, что она мало кому известна. В то же время это довольно полезная функция. Она принимает список выражений или столбцов и возвращает первое значение, которое окажется не пустым. Ее общий синтаксис следующий:

Coalesce(выражение, выражение, …)

##### Функция Nullif()

Иногда пустое значение нужно создать на месте заменяющего его суррогатного. Если база данных заполнена значениями n/a, - или пустыми строками там, где должны находиться пустые значения, вы можете воспользоваться функцией nullif() и расчистить базу данных.

Функция nullif() принимает два аргумента. Если они равны, то возвращается пустое значение, в противном случае возвращается первый параметр.

### *Использование объединений*

Язык SQL содержит множество типов объединений, определяющих, как именно отбираются строки из разных концов объединения. В перечислены эти типы объединений.

**Табл. 17. Типы объединений**

|  |  |
| --- | --- |
| Тип | Определение |
| Внутреннее объединение | Включает только соответствующие строки |
| Левое внешнее объединение | Включает все строки из левой таблицы, независимо от того, существуют ли в правой таблице соответствующие им, а также связанные строки правой таблицы |
| Правое внешнее объединение | Включает все строки из правой таблицы, независимо от того, существуют ли в левой таблице соответствующие им, а также соответствующие строки левой таблицы |
| Полное внешнее объединение | Включает все строки из обеих таблиц, независимо от того, существуют ли соответствия между ними |
| Тета-объединение | Соответствия между строками ищутся с использованием неравенств (<, >, <=, >= и <>) |
| Перекрестное объединение | Декартово произведение – соответствие между каждой строкой первого и второго источника данных, без каких бы то ни было условий и ограничений |

#### Внутренние объединения

Внутренние объединения являются самыми распространенными. Внутренние объединения возвращают только те строки, которые соответствуют друг другу в двух наборах данных. Внутреннее объединение получило свое название по той причине, что оно извлекает данные из внутренней части пересечения двух множеств данных.

Внутренние объединения легко создать в утилите Management Studio с помощью графического инструмента QueryDesigner. После того, как две таблицы помещены на панель диаграммы с помощью функции AddTable или метода перетаскивания из списка таблиц, объединение создается автоматически, если эти таблицы содержат общие поля. Если вы не собираетесь использовать это объединение, то его придется удалить вручную.

В конструкторе запросов для каждого типа объединения используется свой символ. Символ, используемый для внутреннего объединения, имеет вид ромбика.

В коде SQL объединения определяются в предложении FROM инструкции SELECT. Ключевое слово JOIN идентифицирует вторую таблицу, a ON – поля, на которых строится поиск соответствия между таблицами. Внутреннее объединение используется в SQL по умолчанию, поэтому ключевое слово INNER является необязательным.

SELECT \*

FROM Table1

[INNER] JOIN Table2

ON Table1.column = Table2.column

В общем случае инструкция SELECT может обращаться к 255 источникам данных. Так как SQL является декларативным языком, порядок источников данных в инструкции SELECT не имеет значения. Объединения могут быть сформированы в ней во множество цепочек, даже кольцевых (например, A объединяется с B; B – с C; C – с A).

#### Внешние объединения

В то время как внутренние объединения содержат только пересечения двух множеств данных, внешние объединения расширяют это множество за счет несоответствующих данных левого или правого набора данных.

Внешние объединения решают существенную проблему для множества запросов, включая в результат данные, независимо от их соответствия.

Некоторые строки в результирующем наборе данных, сформированном внешним объединением, будут выглядеть точно так же, как строки из внутреннего объединения, - они будут содержать столбцы из каждого источника. В то же время некоторые строки, не нашедшие соответствия на противоположной стороне объединения, будут содержать только столбцы внешней таблицы – столбцы противоположной стороны будут содержать пустые значения.

При создании запросов в конструкторе вы можете в любой момент изменить тип запроса с внутреннего, принятого по умолчанию, на внешний. Это можно сделать с помощью контекстного меню, а также окна свойств объединения. Конструктор запросов отлично справляется с демонстрацией типа объединения, используя для этого специальные символы.

В коде SQL внешнее объединение объявляется с помощью ключевых слов left outer или right outer перед ключевым словом join (технически ключевое слово outer необязательное).

SELECT \*

FROM Table1

LEFT | RIGHT [OUTER] JOIN Table2

ON Table1.column = Table2.column

#### Внешние объединения и необязательные внешние ключи

Внешние объединения часто задействуют, если вторая таблица имеет ограничения внешнего ключа, влияющие на первую таблицу, и допускают пустые значения в столбце внешнего ключа. Наличие этого необязательного внешнего ключа предполагает, что если вторичная строка обращается к первичной строке, то первичная должна существовать. В то же время допустимо, чтобы вторичная строка воздерживалась от обращения к первичной таблице.

Еще одним примером использования необязательного внешнего ключа является столбец приоритетов или предупреждений. Однако в данном случае он должен указывать на допустимую строку в таблице приоритетов.

Пример:

SELECT OrderNumber, OrderPriorityName

FROM dbo.[Order]

LEFT OUTER JOIN dbo.OrderPriority

ON [Order].OrderPriorityID=OrderPriority.OrderPriorityID

Данное левое внешнее объединение извлекает все заказы и приоритеты, соответствующие им (если таковые имеются).

#### Полные внешние объединения

Полное внешнее объединение возвращает все данные из обоих наборов независимо от их пересечения. Функционально такое объединение идентично слиянию с требованием уникальности правого и левого внешних объединений.

В реальной жизни ссылочная целостность уменьшает потребность в полных внешних объединениях, так как любая строка из вторичной таблицы должна иметь соответствие в первичной таблице (в зависимости от обязательности внешнего ключа), так что чаще всего вполне достаточно левого внешнего объединения. Полные внешние объединения особенно полезны для очистки данных, не имеющих ограничений, поскольку отфильтровывают недопустимые значения.

Следующий пример иллюстрирует такую ситуацию. Таблица Two является подчиненной и имеет внешний ключ, ссылающийся на таблицу One. В ней не существует ограничений на внешний ключ, поэтому могут появляться некоторые несоответствия, которые легко выявить с помощью полного внешнего объединения:

CREATE TABLE dbo.One(

OnePK INT,

Thing1 VARCHAR(15)

)

CREATE TABLE dbo.Two(

TwoPK INT,

OnePK INT,

Thing2 VARCHAR(15)

)

Полное внешнее объединение вернет все строки из обеих таблиц, независимо от того, найдено ли для них соответствие или нет:

SELECT Thing1, Thing2

FROM dbo.One

FULL OUTER JOIN dbo.Two

ON One.OnePK=Two.OnePK

Результат выполнения запроса:

|  |  |
| --- | --- |
| Thing1 | Thing2 |
| NULL | Plane |
| New Thing | Train |
| Red Thing | Car |
| NULL | Cycle |
| Blue Thing | NULL |
| Old Thing | NULL |

Как было показано в примере, полное внешнее объединение является прекрасным инструментом поиска всех данных, включая некорректные.

#### Помещение во внешние объединения условий

Если речь идет о внутренних объединениях, условие производит тот же эффект, будучи помещенными в предложение JOIN или WHERE, однако это утверждение не относится к внешним объединениям. Когда условие находится в предложении JOIN, SQL Server включает в результат все строки из внешней таблицы, а затем использует условие для включения строк из вторичной таблицы. Когда ограничение помещено в предложение WHERE, выполняется объединение, а затем условие применяется к объединенным строкам.

Следующие два примера демонстрируют эффект от помещения условия в разные предложения.

В первом запросе левое внешнее объединение включает все строки из таблицы One, а затем объединяет их с теми строками из таблицы Two, для которых значения поля OnePK в обеих таблицах равно Thing1 равно New Thing.

SELECT Thing1, Thing2

FROM dbo.One

LEFT OUTER JOIN dbo.Two

ON One.OnePK=Two.OnePK

AND One.Thing1=’New Thing’

Второй запрос выполняет левое внешнее объединение. После этого предложение WHERE применяет к этому результату ограничение.

SELECT Thing1, Thing2

FROM dbo.One

LEFT OUTER JOIN dbo.Two

ON One.OnePK=Two.OnePK

WHERE One.Thing1=’New Thing’

#### Собственные объединения

Собственными называют такие объединения, в которых таблица ссылается сама на себя. Этот тип отношений часто используют для извлечения данных из возвратных (также называемых рекурсивными) связей. Примером может служить база данных производства строительных материалов (выпускаемый материал ссылается на материалы своих компонентов) и база данных сотрудников предприятия (начальник ссылается на своих подчиненных).

В следующем примере ссылка на таблицу dbo.Person выполняется с помощью именованного диапазона 'Mother'.

Следующий запрос поможет выявить детей Одри Халлоуэй:

SELECT Person.PersonID, Person.FirstName,

Person.MotherID, Mother.PersonID

FROM dbo.Person

JOIN dbo.Person Mother

ON Person.MotherID = Mother.PersonID

WHERE Mother.LastName = ‘Halloway’

AND Mother.FirstName = ‘Audry’

#### Перекрестные (неограниченные) объединения

Перекрестные объединения, также называемые неограниченными, являются реализацией реляционного произведения двух исходных таблиц. При отсутствии условия объединения результирующий набор данных будет содержать все возможные комбинации строк двух источников. Каждая строка исходного набора 1 будет сопоставляться с каждой строкой исходного набора 2. Например, если первый набор данных содержит пять строк, а второй – четыре строки, то результирующий набор данных будет содержать двадцать строк. Такой тип результирующего множества называют декартовым произведением.

В коде этот тип объединения определяется ключевыми словами cross join при отсутствии условия ON.

SELECT Thing1, Thing2

FROM dbo.One

CROSS JOIN dbo.Two

Результатом такого объединения без ограничений будут все строки таблицы One, связанные с каждой строкой таблицы Two.

#### Тета-объединения (Θ-объединения)

Тета-объединения (обозначаемые греческой буквой Θ) – это объединения, основанные на условиях неравенства. В реляционной теории все условные операторы (=, <, <=, >, >= и <>) называют Θ-операторами. Несмотря на то что с теоретической точки зрения условия равенства также используют Θ-оператор, в теории баз данных только объединения, использующие в своих условиях другие операторы, называют Θ-объединениями.

Θ-условие можно создать в конструкторе запросов Management Studio с помощью диалогового окна свойств объединения.

Θ-объединения часто комбинируют со множеством условных объединений, использующих неключевые столбцы.

#### Объединения с множеством условий

Если объединения – это не что иное, как условие связывания двух наборов данных, то имеет смысл использовать в них множество условий. На практике объединения с множеством условий идут рука об руку с Θ-объединениями; без возможности использовать первые вторые практически утратили бы свое значение.

Условия объединения могут ссылаться на любую таблицу, перечисленную в предложении FROM, и это позволяет создавать интересные трехсторонние объединения. Пример:

SELECT \*

FROM A

JOIN B

ON A.col = B.col

JOIN C

ON B.col = C.col

AND A.col = C.col

#### Неключевые объединения

Объединения не ограничены только первичными и внешними ключами – в них можно сопоставлять строки двух наборов данных по любым столбцам, разумеется, если эти столбцы имеют совместимые типы данных.

#### Использование слияний

Слияние функционально отличается от объединения. В терминах реляционной алгебры слияние является сложением, в то время как объединение – умножением. Вместо расширения строк по горизонтали, как это делает объединение, слияние накладывает друг на друга несколько результирующих наборов данных, формируя одну длинную таблицу.   
При проектировании запросов слияния нужно следовать некоторым правилам.

* Имена столбцов или псевдонимов должны быть определены в первой инструкции SELECT.
* Все инструкции SELECT должны иметь одинаковое число столбцов, и каждая линейка столбцов должна использовать типы данных из одного семейства.
* В инструкции SELECT можно добавлять выражения, определяющие источник строк, если данный столбец включен во все инструкции SELECT.
* Слияния можно использовать как часть инструкции SELECT INTO, однако ключевое слово INTO должно находиться в первой инструкции SELECT.
* Если для инструкции SELECT по умолчанию определено все множество строк (ALL) и не определено иное, то для слияния можно утверждать обратное. Результатом слияния являются уникальные строки. Если вы хотите изменить этот режим, следует явно указать ключевое слово ALL.
* Предложение ORDER BY сортирует результаты всех инструкций SELECT, однако при этом использует имена столбцов из первой инструкции SELECT.

В следующем запросе слияния предложение ORDER BY обращается к столбцу Thing1 первой инструкции SELECT:

SELECT OnePK, Thing1, 'из таблицы One' as Source

FROM dbo.One

UNION ALL

SELECT TwoPK, Thing2, ‘из таблицы Two’

FROM dbo.Two

ORDER BY Thing1

#### Слияние пересечения

Слияние пересечения ищет строки, общие для обоих наборов данных. Внутреннее объединение ищет соответствие по горизонтали, а слияние пересечения – по вертикали.

Пример:

SELECT Thing1

FROM dbo.One

INETERSECT

SELECT Thing2

FROM dbo.Two

ORDER BY Thing1

#### Слияние разности/Except

Слияние разности аналогично слиянию пресечения, однако ограничение HAVING выявляет только те строки, которые существуют только в одном из двух наборов данных.

Слияние разности аналогично запросу на разность множеств в том, что ищет все строки, которые присутствуют в одном наборе данных, но отсутствуют в другом. В то же время как запрос на разность

множеств в том, что ищет все строки, которые присутствуют в одном наборе данных, но отсутствуют в другом. В то время как запрос на разность множеств интересуется только условиями объединения (обычно между первичным и внешним ключами) и объединяет строки по горизонтали, слияние разности смотрит на строки в целом (точнее, на все столбцы, участвующие в инструкциях слияния SELECT) по вертикали.

SQL Server 2005 для выполнения слияния разности использует ключевое слово EXCEPT стандарта ANSI:

SELECT Thing1

FROM dbo.One

EXCEPT

SELECT Thing2

FROM dbo.Two

ORDER BY Thing1

### *Модификация данных*

#### Вставка данных

Язык SQL предлагает четыре формы инструкций INSERT и SELECT INTO как основные методы вставки данных. В то время как простые методы реализуют вставку всего одной строки данных, более сложные получают результаты от вложенных инструкций SELECT и создают из результатов таблиц.

**Табл. 18. Формы операций вставки**

|  |  |
| --- | --- |
| Форма вставки | Описание |
| INSERT/VALUES | Вставляет одну строку значений. Обычно используется для поддержки интерактивного ввода данных пользователем |
| INSERT/SELECT | Вставляет в таблицу результирующий набор данных вложенного подзапроса |
| INSERT/EXEC | Вставляет в таблицу результаты хранимой процедуры. Обычно используется в сложных задачах манипулирования данными |
| INSERT DEFAULT | Создает новую строку с применением всех значений, заданных по умолчанию. Обычно используется для вставки общепринятых полей строк данных |
| SELECT INTO | Создает новую таблицу из результатирующего набора данных инструкции SELECT |

Каждую из этих форм инструкции вставки данных лучше применять к конкретной задаче или форме извлечения внешних данных.

#### Вставка одной строки значений

Простейший и самый непосредственный метод вставки данных заключается в использовании инструкции INSERT … VALUES. Так как эта форма принимает только один набор значений, она ограничена вставкой в таблицу только одной строки. Интерфейс пользователя имеет тенденцию принимать ввод только одной строки данных, так что этот метод считается наиболее предпочтительным для использования с формами.

INSERT [INTO] владелец.таблица [(столбец, …)]

VALUES (значение, …)

Создать инструкцию INSERT … VALUES совсем не сложно, но в то же время существует несколько вариантов. Ключевое слово INTO является необязательным и поэтому часто игноририруется. Это ключевое слово проверяет правильность столбцов в списке инструкции INSERT, а также соответствие типа перечисляемых значений типу вставляемых столбцов.

Когда значения вставляются в новую строку, каждое значение соответсвует некоторому столбцу. Столбцы в списке могут перечисляться в любом порядке (не обязательно совпадающем с порядком столбцов в таблице) – главное, чтобы и значение имели тот же порядок.

Потенциально можно форсировать в инструкции INSERT вставку значений по умолчанию, даже не зная об их существовании. Для этого в списке столбцов/значений используют ключевое слово DEFAULT; при этом SQL Server запоминает указанное значение.

#### Вставка результирующего набора данных инструкции SELECT

Данные можно переместить из результирующего набора данных в таблицу с помощью инструкции INSERT … SELECT. Реальная сила этого метода, а также его гибкость заключается в том, что сама инструкция SELECT может извлечь данные практически из любого места и адаптировать их к текущим потребностям. Так как инструкция SELECT может вернуть бесконечное число строк, все они могут быть вставлены в таблицу. Синтаксис этой инструкции следующий.

INSERT [INTO] владелец.таблица

SELECT столбцы

FROM источники\_данных

[WHERE условия]

Как и в инструкции INSERT … VALUES, столбцы должны быть упорядочены и иметь соответствующие типы данных. Если необязательный список столбцов опускается, все столбцы должны заполняться в том порядке, в котором объявлены при создании таблицы (за исключением столбца идентичности).

#### Вставка результирующего набора данных из хранимой процедуры

Форма INSERT … EXEC инструкции вставки использует результаты выполнения хранимой процедуры для их вставки в таблицу. В данном случае можно использовать все возможности языка T-SQL. Базовая функция вставки является такой же, как и во всех ее остальных формах. Порядок столбцов в списке инструкции INSERT и в результатах хранимой процедуры должен быть одинаковым. Базовый синтаксис инструкции:

INSERT [INTO] владелец.таблица [(столбцы)]

EXEC хранимая\_процедура параметры

Создание строки со значениями по умолчанию

Язык SQL имеет особую форму инструкции INSERT, которая создает новую строку, содержащую только значения столбцов по умолчанию. Единственным параметром такой инструкции является имя таблицы, при этом значения и имена столбцов не требуются, не принимаются во внимание. Синтаксис такой инструкции простой:

INSERT владелец.таблица DEFAULT VALUES

#### Создание таблицы в процессе вставки данных

Последний метод вставки данных использует одну из вариаций инструкции SELECT. Используемый в ней параметр INTO принимает результирующий набор данных от инструкции SELECT и создает на его основе новую таблицу. Форму SELECT … INTO часто используют в операциях преобразования данных и в утилитах, которые должны работать с массой различных структур таблиц. Полный синтаксис этой формы включает все параметры инструкции SELECT, поэтому здесь приводится усеченная версия:

SELECT столбцы

INTO новая\_таблица

FROM источники\_данных

[WHERE условия]

Структура данных созданной таблицы может не соответствовать в точности исходной, поскольку она основана на комбинации столбцов исходной таблицы и результирующего набора инструкции SELECT.

#### Обновление данных

Инструкция UPDATE языка SQL предельно проста. Она способна изменить значение всего одной ячейки, а также всех столбцов всех строк таблицы. Однако предложение FROM позволяет таблице стать частью составного источника данных и использовать всю силу инструкции SELECT.

Вот как работает инструкция UPDATE:

UPDATE dbo.таблица

SET столбец = значение/выражение/столбец,

столбец = значение…

[FROM источники\_данных]

[WHERE условия]

Инструкция UPDATE может изменить значения множества строк, но только одной таблицы. Ключевое слово SET используется для преобразования данных в новые значения. Эти новые значения могут быть константами, переменными, выражениями и даже столбцами другого источника данных, имеющегося в предложении FROM инструкции UPDATE.

#### Удаление данных

Инструкция DELETE в своей простейшей форме удаляет все строки таблицы. Так как эта инструкция работает с целыми строками, ей не требуется явного указания столбцов. Первое предложение FROM не является обязательным, равно как и второе предложение WHERE. Но несмотря на то, что предложение WHERE не обязательно, оно прежде всех остальных заботится о том, какие именно строки будут удаляться из таблицы. Вот сокращенный синтаксис инструкции DELETE:

DELETE [FROM] владелец.таблица

[FROM источники\_данных]

[WHERE условия]

## **Представления**

Представлением называют сохраненную инструкцию SQL, на которую можно сослаться как на источник данных в запросе, так же как на подзапрос. Представление не может быть выполнено самостоятельно; оно должно использоваться в запросе.

Подобно любому запросу SQL, представления обращаются к данным, хранимым в таблицах. Основной задачей представлений является отображение данных в удобном формате.

### *Работа с представлениями*

В утилите SQL Server Management Studio представления можно создавать, редактировать, выполнять и вставлять в другие запросы.

Поскольку представление является ничем иным, как сохраненной инструкцией SELECT, его создание начинается с проектирования этой инструкции. Инструкция SELECT, если она является корректной, может быть вырезана и вставлена в представление практически из любого инструмента.

В утилите Management Studio представления перечислены в собственном узле в каждой базе данных.

Команда New View в контекстном меню позволит запустить конструктор запросов в режиме создания представлений.

Конструктор представлений работает в конструкторе запросов Management Studio. Фактический код SQL отображается и редактируется на панели SQL. Столбцы в представление можно добавлять на панелях Diagram, Grid и SQL. Функция добавления таблиц доступна в контекстном меню, а также на панели инструментов. Здесь можно добавлять таблицы, другие представления, синонимы и табличные функции.

Представлениями можно управлять в редакторе запросов, выполняя сценарии SQL, которые используют команды языка DDL: CREATE, ALTER и DROP. Основной синтаксис создания представления следующий:

CREATE dbo.имя\_представления

AS

инструкция\_SELECT

Когда представление создано, инструкцию SELECT можно с легкостью отредактировать с помощью команды ALTER:

ALTER dbo.ViewName

AS

SQL Select Statement

## **Хранимые процедуры**

Одним из самых популярных методов переноса обработки ближе к данным является создание хранимых процедур. В них используются те запросы и пакеты инструкций T-SQL. Точно так же, как запрос может быть сохранен в представлении, пакет инструкций может быть сохранен под именем хранимой процедуры, одновременно проходя процесс компиляции.

Как серверные программы хранимые процедуры имеют ряд преимуществ.

* Хранимые процедуры хранятся в компилированном виде, поэтому выполняются быстрее, чем пакеты или запросы.
* Выполнение обработки данных на сервере, а не на рабочей станции, значительно снижает нагрузку на локальную сеть.
* Хранимые процедуры имеют модульный вид, поэтому их легко внедрять и изменять. Если клиентское приложение вызывает хранимую процедуру для выполнения некоторой операции, то модификация процедуры в одном месте влияет на ее выполнение у всех пользователей.
* Хранимые процедуры можно рассматривать как важный компонент системы безопасности базы данных. Если все клиенты осуществляют доступ к данным с помощью хранимых процедур, то прямой доступ к таблицам может быть запрещен, и все действия пользователей будут находиться под контролем.

### *Инструкции CREATE, ALTER и DROP*

Хранимые процедуры управляются посредством инструкций языка определения данных (DDL) CREATE, ALTER и DROP. Инструкция CREATE должна быть первой в пакете; терминатор пакета завершает создание хранимой процедуры. В следующем примере создается простая хранимая процедура, которая извлекает данные из таблицы ProductCategory.

CREATE PROCEDURE CategoryList

AS

SELECT ProductCategoryName, ProductCategoryDescription

FROM dbo.ProductCategory;

RETURN;

Инструкция DROP удаляет хранимую процедуру из базы данных. Инструкция ALTER изменяет содержимое хранимой процедуры. Для внесения изменения предпочтительнее использовать инструкцию ALTER, а не комбинацию инструкций удаления и создания, так как последний метод удаляет все разрешения.

### *Возвращение набора записей*

Как пакет может вернуть набор записей из запроса SELECT, так и хранимая процедура может вернуть набор записей из запроса.

### *Компиляция хранимых процедур*

Компиляция хранимых процедур выполняется автоматически при первом из запуске, после чего скомпилированный код сохраняется в памяти (точнее, SQL Server создает план выполнения запросов и программного кода хранимой процедуры, после чего сохраняется в памяти).

### *Передача данных в хранимые процедуры*

Хранимая процедура становится только более ценной, если использует параметры. Хранимые процедуры SQL Server могут иметь до 2100 единиц входных и выходных параметров.

#### Входные параметры

В инструкции CREATE PROCEDURE можно перечислить параметры, передаваемые хранимой процедуре, указав их непосредственно после ее имени. Каждый из параметров должен начинаться с символа @. Для хранимой процедуры он является локальной переменной. Как и все локальные переменные, параметры должны объявляться с допустимыми типами данных. При вызове хранимой процедуры должны также указываться все параметры (если, конечно, некоторые из них не имеют значений, заданных по умолчанию).

#### Значения параметров, заданные по умолчанию

При вызове хранимой процедуры вы обязаны указать все ее параметры, кроме тех, для которых были определены значения по умолчанию. В определении хранимой процедуры значения по умолчанию определяются добавлением к имени параметра знака равенства и значения:

CREATE PROCEDURE имя\_процедуры (

@переменная тип\_данных = значение\_по\_умолчанию

);

### *Получение данных из хранимой процедуры*

SQL Server предлагает четыре способа получения данных из хранимой процедуры. Пакет может вернуть данные из инструкции SELECT или команды raiseerror. Хранимые процедуры используют эти методы пакетов и добавляют к ним два собственных: выходные переменные и команду output.

#### Выходные параметры

Выходные параметры позволяют хранимой процедуре возвращать данные вызывающей клиентской программе. При этом указание ключевого слова output обязательно как при определении процедуры, так и при ее вызове. В самой хранимой процедуре выходные параметры являются локальными переменными. В вызывающей процедуре или пакете выходные переменные должны быть предварительно определены, чтобы получить результирующие значения. Когда выполнение хранимой процедуры завершается, текущее значение параметра передается локальной переменной вызывающей программы.

#### Использование команды RETURN

Команда return позволяет завершить выполнение процедуры в ее середине и вернуть значение вызывающему пакету или клиенту.

Возвращенное значение 0 указывает на успешное выполнение процедуры и установлено по умолчанию. Компания Microsoft зарезервировала значения от -99 до -1 для служебного пользования. Разработчиком для возвращения состояния ошибки пользователю рекомендуется использовать значения – 100 и меньше.

## **Триггеры**

Триггеры представляют собой специальные хранимые процедуры, прикрепленные к событиям таблиц. Их невозможно выполнить вручную; их запуск является откликом на события вставки, обновления и удаления данных из таблицы.

В SQL Server 2005 существует два типа триггеров транзакций: INSTED OF (триггер замены операции) и AFTER (триггер, выполняющийся сразу после операции). Эти типы отличаются своим назначением, временем выполнения и производимым эффектом.

**Табл. 19. Сравнение типов триггеров**

|  |  |  |
| --- | --- | --- |
|  | **Триггер INSTEAD OF** | **Триггер AFTER** |
| Инструкция DML | Автоматически откатывается | Выполняется, если триггер сам не откатит транзакцию |
| Время выполнения | Перед ограничениями первичного и внешнего ключей | После выполнения транзакции, но перед ее подтверждением |
| Количество возможных событий таблицы | Одно | Несколько |
| Можно ли применять к представлениям? | Да | Нет |
| Допустима ли вложенность? | В зависимости от параметров сервера | В зависимости от параметров сервера |
| Допустима ли рекурсивность? | Нет | В зависимости от параметров сервера |

### *Порядок выполнения транзакций*

Создание триггеров требует четкого понимания общего порядка выполнения транзакций, в противном случае может возникнуть конфликт между ограничениями и триггерами.

Любая транзакция проходит несколько проверок и программных кодов в следующем порядке.

1. Проверка Identity Insert.
2. Ограничение допустимости пустых значений.
3. Проверка типа данных.
4. Выполнение триггера INSTEAD OF. Если такой триггер существует, то выполнение инструкции DML останавливается в этой точке. Триггеры INSTEAD OF не могут быть рекурсивными. Таким образом, если триггер, запущенный событием некоторой инструкции, снова выполняет ту же инструкцию (INSERT, UPDATE или DELETE), во второй раз его присутствие игнорируется.
5. Ограничение первичного ключа.
6. Ограничение проверки.
7. Ограничение внешнего ключа.
8. Выполнение инструкции DML и обновление журнала транзакций.
9. Выполнения триггера AFTER.
10. Подтверждение транзакции.
11. Запись в файл данных.

На основе знания порядка выполнения транзакций можно сделать несколько замечаний, касающихся проектирования триггеров.

* Триггер AFTER выполняется после всех ограничений. По этой причине он не может скорректировать данные, поэтому данные должны пройти проверки ограничениями, включая ограничение внешнего ключа.
* Триггер INSTEAD OF может помочь обойти проблемы, связанные с внешним ключом, но не с допустимостью пустых значений, типом данных или столбцом идентичности.
* Триггер AFTER может принимать во внимание, что данные уже прошли все встроенные проверки целостности данных.
* Триггер AFTER выполняется перед подтверждением транзакции. Таким образом, если данные недопустимы, он имеет возможность откатить транзакцию.

### *Создание триггеров*

Триггеры создаются и модифицируются с помощью стандартных команд языка DDL CREATE, ALTER и DROP следующим образом:

CREATE TRIGGER имя\_триггера ON имя\_таблицы

AFTER Insert, Update Delete

AS

программный\_код\_триггера

### *Триггеры AFTER*

Таблица может иметь несколько триггеров AFTER для каждого из трех своих событий (вставка, обновление и удаление). Триггеры AFTER применимы только к таблицам.

Традиционный триггер является триггером AFTER, который запускается после выполнения транзакции, но до ее подтверждения. Эти триггеры могут пригодиться для следующих операций:

* сложная проверка данных;
* поддержка сложных правил бизнес-логики;
* запись журнала аудита данных;
* обслуживание измененных столбцов данных;
* реализация пользовательских проверок целостности данных и каскадных удалений.

### *Триггеры INSTEAD OF*

Триггер INSTEAD OF заменяет транзакцию (т.е. выполняется вместо нее). Это подобно тому, как если бы триггер автоматически откатил транзакцию, для которой был создан.

Каждая таблица ограничена возможностью использования только одного триггера подстановки для каждого из своих событий. В то же время триггеры INSTEAD OF могут применяться к представлениям точно так же, как к таблицам.

Триггеры INSTEAD OF особенно полезны, когда заранее известно, что инструкция DML, запустившая триггер, почти наверняка будет отменена, а вместо нее должна быть реализована некоторая логика. Приведем примеры.

* Инструкция DML пытается обновить необновляемое представление. В этом случае триггер INSTEAD OF обновляет таблицы, на которых построено данное представление.
* Инструкция DML пытается напрямую обновить таблицу складских остатков, что недопустимо. В этом случае триггер INSTEAD OF обновляет таблицу складских операций.
* Инструкция DML пытается удалить строку, а триггер INSTEAD OF вместо этого перемещает данную строку в архивную таблицу.

### *Работа с транзакциями*

Запускать триггер могут инструкции DML INSERT, UPDATE и DELETE. Очень важно, чтобы триггер имел доступ к изменениям, выполненным инструкцией DML, чтобы проверить полученные значения или обработать результаты транзакции. SQL Server предлагает четыре способа проверки в теле триггера эффекта произведенного инструкцией DML. Образы Inserted и Deleted содержат наборы до и после выполнения инструкции, а функции updated() и columns\_updated() можно использовать для определения того, на какие столбцы оказала воздействие инструкция DML.

#### Определение состава обновленных столбцов

SQL Server предлагает два метод определения состава обновленных столбцов. Функция update() возвращает значение true для одного столбца, если на него повлияла транзакция DML:

IF UPDATE(ColumnName)

Функция columns\_updated() возвращает карту varbinary наличия изменений в столбцах таблицы. Если конкретный бит имеет значение true, это значит, что столбец был изменен. Результат функции column\_updated() можно сравнить с целочисленными или двоичными данными с помощью любой битовой операции, чтобы определить, был ли изменен конкретный столбец.

#### Логические таблицы Inserted и Deleted

SQL Server позволяет программному коду в теле триггера получить доступ к эффекту от транзакции, которая вызвала триггер. Логические таблицы Inserted и Deleted представляют собой образы данных, доступные только для чтения. Эти таблицы можно рассматривать как представления журнала транзакций.

Таблица Deleted содержит строки в состоянии до применения инструкции DML, а таблица Inserted – после применения.

**Табл. 20. Таблицы Inserted и Deleted**

|  |  |  |
| --- | --- | --- |
| **Инструкция DML** | **Таблица Inserted** | **Таблица Deleted** |
| INSERT | Вставленные строки | Пустая |
| UPDATE | Строки базы данных после обновления | Строки базы данных до обновления |
| DELETE | Пустая | Строки, подлежащие удалению |

Таблицы Inserted и Deleted имеют очень ограниченную область определения. Их можно увидеть только внутри триггера; хранимые процедуры, вызываемые триггером, не имеют доступа к этим таблицам.

## **Курсор**

По своей сути курсор является указателем на одну строку данных. Для прохождения по набору данных до его конца обычно используют цикл while. SQL Server поддерживает стандартный синтаксис ANSI SQL-92 и расширенный синтаксис T-SQL, предлагающий дополнительные возможности.

### *Пять этапов жизни курсора*

Курсор создает результирующий набор данных на основе инструкции SELECT, а затем проходит по нему построчно. Пятью этапами жизни курсора являются следующие.

1. Объявление курсора определяет тип и режим его работы, а также описывает инструкцию SELECT, поставляющую ему данные. При объявлении курсора никакие данные не извлекаются. Это единственный случай, когда инструкция declare не требует амперсанда. Курсор SQL-92 объявляется с помощью инструкции cursor for:  
   DECLARE имя\_курсора CURSOR  
   FOR инструкция\_SELECT  
   FOR параметры\_курсора  
   Расширенный курсор T-SQL объявляется аналогичным образом:  
   DECLARE имя\_курсора CURSOR параметры\_курсора  
   FOR инструкция\_SELECT
2. При открытии курсора извлекаются данные, которыми он и заполняется:  
   OPEN имя\_курсора
3. Курсор перемещается к следующей строке и заполняет значения ее столбцов локальными переменными (эти переменные должны быть предварительно объявлены):  
   FETCH [направление] имя\_курсора INTO @переменная1, @переменная2  
   По умолчанию команда FETCH перемещает курсор к следующей строке (направление NEXT), однако при желании можно переместить курсор к предыдущей (PRIOR), первой (FIRST) и последней (LAST) строке. Также с помощью этой команды можно переместить курсор к строке с некоторым абсолютным номером (ABSOLUTE n) или сместить относительно текущей позиции на определенное расстояние (RELATIVE n). Проблема последнего подхода состоит в том, что в реляционной базе данных номер строки не имеет определенного смысла. Если в коде требуется перейти к конкретной строке, чтобы получить логический результат, то это должно быть заранее предусмотрено в модели базы данных.
4. Закрытие курсора снимает блокировку данных, но сохраняет инструкцию SELECT. Курсор впоследствии может быть открыт в той же точке:  
   CLOSE имя\_курсора
5. Демонтаж курсора высвобождает отведенную под него память и аннулирует его определение:  
   DEALLOCATE имя\_курсора

### *Управление курсором*

Так как курсор проходит по набору данных построчно, код T-SQL требует команд постоянного смещения курсора. Для управления этим циклическим процессом язык T-SQL предлагает две глобальные переменные, содержащие информацию о состоянии курсора.

Переменная @@cursor\_rows возвращает общее количество строк в курсоре. Если курсор заполняется асинхронно, то в этой переменной будет содержаться отрицательное значение.

Существенной для управления курсором является глобальная переменная @@fetch\_status. Она отчитывается о состоянии курсора после последней команды FETCH. Это состояние важно для управления перемещения курсора и оценка, достиг ли он одного из концов набора данных. Возможные значения переменной @@fetch\_status свидетельствуют о следующем.

* 0 – последняя операция FETCH успешно извлекла строку.
* 1 – последняя операция FETCH достигла конца набора данных.
* 2 – строка, к которой переместился курсор, оказалась недоступной; он была удалена.

Комбинирование переменной @@fetch\_status с оператором while позволяет создать циклы, позволяющие успешно перемещаться по строкам набора данных.

Обычно в пакетах создается курсор, после чего выполняется первая команда FETCH и начинается цикл while, который продолжается до тех пор, пока не будет достигнут конец набора данных. В верхней части цикла проверяется значение переменной @@fetch\_status для определения, создан ли курсор.

## **Выполнение массовых операций**

В работе часто требуется быстро выполнить копирование больших объемов информации. Если несколько сотен мегабайт информации должны переместиться в базы данных SQL Server за ограниченный промежуток времени, то на помощь придут массовые операции.

В SQL Server массовые операции выполняются в обход журнала транзакций, направляя данные непосредственно в базу данных.

Массовые операции можно выполнить с помощью утилиты командной строки BCP, инструкции bulk insert языка T-SQL, а также службы интеграции.

### *Команда bulk insert*

Команда bulk insert может использоваться в сценариях T-SQL и хранимых процедурах для импорта данных в SQL Server. В параметрах этой команды указываются таблица, получающая данные, путь к исходному файлу и параметры.

Первое что нужно узнать bulk insert, - это то, что каждый столбец источника данных вставляется непосредственно в таблицу назначения, используя отображения «один к одному». Первый столбец источника вставляется в первый столбец таблицы приемника, второй – во второй и т.д. Если в таблице приемника слишком много столбцов, то лишние игнорируются. Если же в ней столбцов меньше, чем нужно, то из лишних данных источника получится настоящее месиво.

### *Утилита BCP*

Утилита BCP (эта аббревиатура расшифровывается как «программа массового копирования») – вариация командной строки операции массового копирования. Она отличается от операции массовой вставки тем, что может как импортировать, так и экспортировать данные. Эта утилита использует многие из параметров операции bulk insert. Ее базовый синтаксис следующий:

BCP таблица\_назначения путь\_к\_файлу\_данных параметры

Так как эта программа является внешней, он требует авторизации для подключения к серверу. Есть два варианта действий: указать в командной строке параметр –P и запрограммировать пароль в пакетном файле или не указывать параметр –P, и тогда система сама запросит у вас ввод пароля.

## **Измерение производительности**

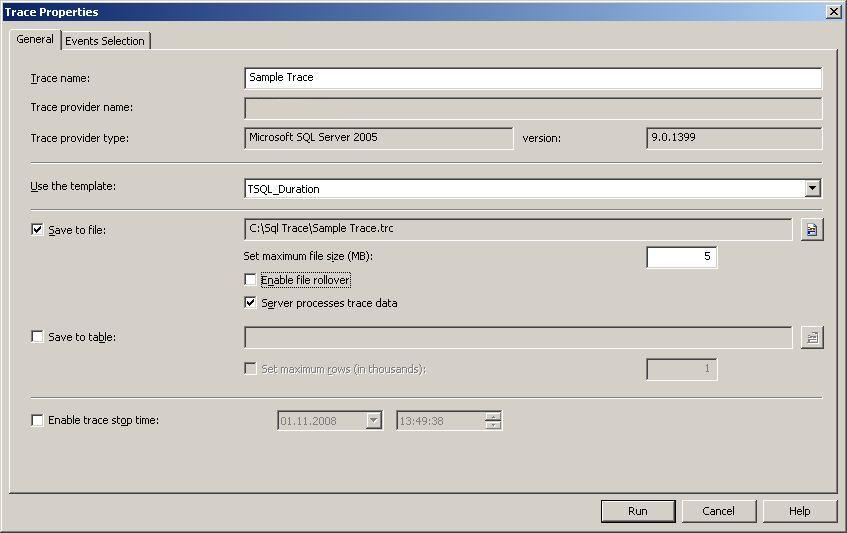
В основе стратегий оптимизации лежат измерения производительности. Без конкретных цифр любая оценка производительности была бы чисто субъективной. Только цифры помогают выявить тенденции и узкие места и учитывать эти показатели в настройке базы данных. Теория оптимизации в основе которой лежат целостные измерения, позволяет проектировать высокопроизводительные базы данных.

Для измерения производительности SQL Server доступно несколько методов.

* В монитор производительности (Perfomance Monitor) системы Windows при установке SQL Server встраивается несколько дополнительных счетчиков, позволяющих собирать информацию о производительности сервера баз данных в журналах для дальнейшего анализа.
* Утилита SQL Server Profiler позволяет с предельной детализацией отслеживать практически все события в сервере базы данных.
* Ядро базы данных возвращает статистику времени выполнения запросов и операций ввода-вывода.
* Пакеты T-SQL и хранимые процедуры могут использовать функцию GetDate() для вычисления времени выполнения команд с точностью до трех миллисекунд и занесения полученных значений в журнал.

### *Использование SQL Server Profiler*

Эта утилита отображает данные о любом количестве детализированных событий SQL Server. Эти события сервера можно просматривать в окне Trace Properties (Рис.7), а также записывать в файл или таблицу для последующего анализа. Для регистрации все событий или их избранного подмножества можно установить фильтры.

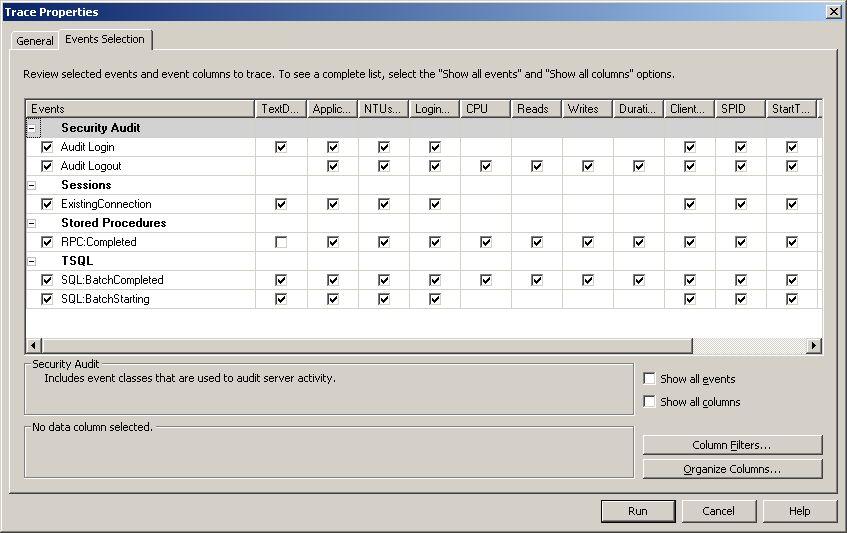


**Рис. 7. В этом окне SQL Server Profiler использован шаблон T-SQL Duration для отображения времени выполнения всех инструкций T-SQL**

Утилиту SQL Server Profiler можно запустить из меню Tools программы Management Studio или непосредственно из папки SQL Server 2005 системного меню Пуск. Для просмотра действий вам необходимо либо определить новую трассировку, либо воспользоваться уже существующим файлом.

#### Определение новой трассировки

Когда создается новая трассировка (либо с помощью команды меню File -> New Trace, либо с помощью кнопки New Trace панели инструментов), создается также новое подключение к SQL Server и открывается диалоговое Trace Properties. Во вкладке General этого окна настраивается трассировка (в частности, имя, местоположение файла и т.п.), а во вкладке Events Selection определяются регистрируемые события, данные и фильтры. Если трассировка запущена, то эти параметры можно просматривать, но не изменять. Конфигурация трассировки может быть сохранена в виде шаблона, чтобы облегчить создание новых трассировок в будущем.



**Рис. 8. Вкладка Events Selection окна Trace Properties позволяет отобрать события, отслеживаемые утилитой Profiler**

Трассировку можно просматривать в реальном времени, однако эти данные могут одновременно записывать в файл или таблицу SQL Server. Это полезно для последующего интеллектуального анализа, сравнения с данными счетчиков монитор системы или для импорта в утилиту Database Engine Tuning Advisor.

Когда показания записываются в файл, для повышения производительности они объединяются в цепочки по 128 Кбайт; аналогично, при записи в таблицу данные группируются по несколько строк.

#### Отбор событий

Во вкладке Events Selection определяется состав действий, выполняемых сервером баз данных, которые будут регистрироваться утилитой Profiler. Подобно монитору производительности, Profiler может отслеживать множество ключевых событий SQL Server. Для упрощения настройки отбора можно использовать шаблоны, предлагаемые по умолчанию.

Для воспроизведения трассировки можно использовать далеко не все события. Например, событие SQL Batch Start может быть воспроизведено, а событие SQL Batch Complete – нет.

В зависимости от событий для трассировки становятся доступными разные данные. Несмотря на то, что столбец данных SPID кажется необязательным, это впечатление обманчиво – он обязателен.

#### Фильтрация событий

Программа Profiler способна собрать для вас такое количество информации, которая легко может в мгновение ока заполнить дисковое устройство. К счастью, предлагаемый программой фильтр поможет ограничить этот массив только интересующими вас данными.

Фильтры используют комбинацию операторов EQUAL и LIKE, зависящую от типа извлекаемых данных. К сожалению, эти фильтры применяются к уже накопленным данным, а данные, собранные для некоторых столбцов, могут оказаться неожиданными. Например, если вы хотите ограничить трассировку только пакетами, работающими с определенными столбцами или таблицами, фильтрация по имени такого объекта работать не будет. В то же время использование фильтра LIKE с символами макроподстановки в столбце text data приведет к тому, что Profiler отберет только те пакеты, которые содержат имя заданной таблицы.

Еще один популярный фильтр, устанавливаемый в Profiler, отсеивает только те пакеты, продолжительность выполнения которых превышает заданное время. Это позволяет отследить все слишком долго выполняемые пакеты.

Установка флажка Exclude system ID позволяет отфильтровать только пользовательские объекты.

#### Использование трассировки

Как только информация о событиях будет собрана, ее можно просмотреть в окне трассировки программы Profiler, несмотря на то, что считается хорошим тоном накапливать эти данные в таблице базы данных. Эти данные можно затем проанализировать, а также выполнить над ними другие действия, доступные для данных обычных таблиц SQL.

После того как файл записан, его можно открыть в SQL Server Profiler и посмотреть данные. Эти данные можно переписать в таблицу для дальнейшего обобщающего анализа с помощью команды File => Save As.

# **Источники**

1. Пол Нильсен. SQL Server 2005. Библия пользователя. ООО «И.Д. Вильямс», 2008. ISBN 978-5-8459-1314-2.
2. Учебный курс Microsoft. Microsoft SQL Server 2005. Реализация и обслуживание. Экзамен 70-431 MCTS.
3. А.Д. Хомоненко, В.М. Цыганков, М.Г. Мальцев. Базы данных. Учебник для высших учебных заведений. Санкт-Петербург, «КОРОНА принт», 2004

# **Список рекомендованной литературы для дополнительного изучения**

1. Мартин Грубер. Понимание SQL.
2. Рэймонд Фрост. Базы данных. Проектирование и разработка.