

# Microsoft SQL Server

## Планирование и архитектура

Источник:           Техническая документация  
Microsoft © 26. янв. 2017

Составитель:       Камнев Георгий  
<[nt.gocha@gmail.com](mailto:nt.gocha@gmail.com)>

Дата составления:   Вт, 31 Январь, 2017

## Оглавление

Архитектура страниц и экстенгов.....	9
Страницы и экстенги.....	9
Страницы.....	9
Поддержка больших строк.....	11
Экстенги.....	11
Управление размещением экстенга и свободным местом.....	12
Управление размещением экстенга.....	12
Отслеживание свободного места.....	13
Управление дисковым пространством, занятым объектами.....	14
Отслеживание измененных экстенгов.....	16
Структура журнала транзакций.....	17
Логическая архитектура журнала транзакций.....	17
Физическая архитектура журнала транзакций.....	18
Контрольные точки и активная часть журнала.....	20
Функционирование контрольной точки.....	20
Действия, приводящие к срабатыванию контрольных точек.....	20
Автоматические контрольные точки.....	21
Активный журнал.....	22
Длительные транзакции.....	23
Транзакции репликации.....	23
Журнал транзакций с упреждающей записью.....	23
Архитектура таблиц и структур данных индекса.....	24
Организация таблиц и индексов.....	24
Организация таблиц.....	24
Секции.....	25
Кластеризованные таблицы, кучи и индексы.....	25
Некластеризованные индексы.....	26
XML-индексы.....	26
Единицы распределения.....	26
Единица распределения IN_ROW_DATA.....	27
Единица распределения ROW_OVERFLOW_DATA.....	28
Единица распределения LOB_DATA.....	28
Пример секции и единицы распределения.....	29
Структуры кучи.....	29
Структуры кластеризованного индекса.....	31
Структуры некластеризованных индексов.....	32
Индексы с включенными столбцами.....	34
Архитектура обработчика запросов.....	34
Обработка инструкций SQL.....	35
Оптимизация инструкций SELECT.....	35
Обработка инструкции SELECT.....	37
Обработка других инструкций.....	37
Рабочие таблицы.....	38
Разрешение представлений.....	38
Использование подсказок с представлениями.....	39
Разрешение индексов для представлений.....	40
Использование подсказок с индексированными представлениями.....	42
Разрешение распределенных секционированных представлений.....	43

Выполнение хранимых процедур и триггеров.....	44
Кэширование и повторное использование плана выполнения.....	45
Удаление планов выполнения из кэша процедур.....	46
Перекомпиляция планов выполнения.....	47
Параметры и повторное использование планов выполнения.....	49
Простая параметризация.....	51
Принудительная параметризация.....	52
Типы данных аргументов.....	54
Рекомендации по использованию принудительной параметризации.....	54
Подготовка инструкций SQL.....	55
Параллельная обработка запросов.....	57
Степень параллелизма.....	58
Пример параллельного запроса.....	60
Параллельные операции с индексами.....	62
Архитектура распределенных запросов.....	63
Улучшенные возможности обработки запросов для секционированных таблиц и индексов.....	65
Новая операция поиска, учитывающая секционирование.....	66
Отображение сведений о секционировании в планах выполнения запросов.....	67
Улучшенные возможности информации о секции.....	67
Отображение сведений о секционировании с помощью других методов Showplan.....	70
Интерпретация планов выполнения для секционированной кучи.....	71
Интерпретация планов выполнения для выровненных соединений.....	71
Стратегия выполнения параллельных запросов для секционированных объектов.....	72
Советы и рекомендации.....	74
Пример.....	74
Архитектура управления памятью.....	76
Архитектура оперативной памяти.....	76
Выделение SQL Server максимального объема памяти.....	77
Адресное пространство процесса.....	78
Динамическое управление памятью.....	80
Параметры настройки min server memory и max server memory.....	80
Требования к объему памяти для хранения объектов SQL Server.....	81
Управление буферами.....	82
Принцип работы управления буфером.....	82
Поддерживаемые возможности.....	83
Операции дискового ввода-вывода.....	84
Длительные запросы операций ввода-вывода.....	84
Причины длительных запросов операций ввода-вывода.....	85
Определение ошибки.....	85
Защита от разрыва страницы.....	86
Защита контрольной суммой.....	86
Считывание страниц.....	86
Запись страниц.....	89
Управление памятью для больших баз данных.....	91
Использование расширений AWE.....	92
Включение поддержки более 4 ГБ физической памяти.....	93
Закрепление страниц в памяти.....	94
Параметр awe enabled.....	94
Увеличение пропускной способности в сетевом приложении до предела.....	94
Активация памяти расширений AWE для SQL Server.....	95
Использование отображаемой памяти расширений AWE с Windows 2000 Server.....	95
Использование отображаемой памяти расширений AWE с Windows Server 2003.....	96

Использование кластера отработки отказа с AWE.....	99
Память с «горячей» заменой.....	99
Основные сведения о неоднородном доступе к памяти.....	100
Основные понятия NUMA.....	101
Сравнение оборудования NUMA с программной архитектурой.....	102
Как SQL Server поддерживает архитектуру NUMA.....	103
Группирование общих ЦП.....	104
Как в SQL Server выполняется сопоставление узлов с программной архитектурой NUMA с узлами оборудования NUMA.....	104
Как назначить соединения для узлов NUMA.....	105
Ограничения, налагаемые версиями SQL Server.....	105
Расширение и сжатие буферного пула в конфигурации с неоднородным доступом к памяти (NUMA).....	105
Распределение памяти.....	105
Выделение памяти при загрузке.....	106
Ограничение памяти определенными узлами.....	106
Освобождение памяти узла.....	106
Внешние страницы.....	107
Наблюдение за использованием локальной и внешней памяти в буферном пуле.....	107
Каждый из узлов контролирует свою память.....	107
Поведение при просмотре таблицы.....	107
Сценарии NUMA.....	107
Полезные сценарии.....	108
А. Нет соответствия портов и NUMA.....	108
Б. Привязка одного порта к нескольким узлам, чтобы обеспечить большую производительность для приоритетного приложения.....	108
В. Привязка нескольких портов к нескольким узлам.....	109
Архитектура задач и потоков.....	109
Работа с расписаниями задач и пакетов SQL Server.....	110
Основные сведения о планировании.....	110
Управление соединениями пользователей и ресурсами рабочих потоков.....	111
Настройка SQL Server на использование волокон.....	111
Принципы работы с расписаниями задач и пакетов SQL Server.....	112
Распределение потоков ЦП.....	113
Использование параметра lightweight pooling.....	113
Выполнение потоков и волокон.....	114
ЦП с поддержкой горячей замены.....	114
Рекомендации по использованию SQL Server на компьютерах, которые имеют более 64 ЦП.....	115
Связывание аппаратных потоков с процессорами.....	115
Управление размером файла журнала транзакций.....	115
Задание максимальной степени параллелизма для операций с индексами.....	115
Задание максимального числа рабочих потоков.....	116
Использование приложений SQL Trace и SQL Server Profiler.....	116
Задание числа файлов данных базы данных tempdb.....	116
Компоненты SQL Server, которые поддерживают более 64 ЦП.....	116
Планирование аварийного восстановления.....	117
Подготовка к аварийному восстановлению.....	117
Модели восстановления SQL Server.....	118
Управление носителем резервной копии.....	119
Выполнение сценария базовой функциональности.....	119
Проверка степени готовности к аварийным ситуациям.....	119

Аудит и предотвращение потенциально катастрофических ошибок, вызываемых действиями пользователей.....	120
Восстановление после сбоя.....	121
Федеративные серверы баз данных.....	122
Основные сведения о федеративных серверах баз данных.....	122
Проектирование федеративных серверов баз данных.....	122
Правила планирования федеративных серверов баз данных.....	122
Симметричные секции.....	122
Асимметричные секции.....	123
Проектирование распределенных секционированных представлений.....	123
Шаблон инструкций SQL, выполняемых приложением.....	124
Связи между таблицами.....	124
Частота использования инструкций SQL применительно к секциям.....	125
Правила маршрутизации инструкций SQL.....	125
Проектирование федеративных серверов баз данных для обеспечения высокого уровня доступности.....	125
Реализация федеративных серверов баз данных.....	126
Создание распределенных секционированных представлений.....	126
Создание таблиц-элементов.....	126
Определение распределенных секционированных представлений.....	127
Правила таблиц.....	128
Правила столбцов.....	128
Правила столбцов секционирования.....	128
Общие правила.....	129
Изменение данных в секционированных представлениях.....	129
Инструкции INSERT.....	130
Инструкции UPDATE.....	131
Инструкции DELETE.....	131
Резервное копирование и восстановление федеративных серверов баз данных.....	131
Таблицы.....	132
Основные сведения о таблицах.....	132
Основы таблиц.....	132
Основы целостности данных.....	132
Значения NULL.....	132
Ограничения, правила, значения по умолчанию и триггеры.....	132
Ограничения.....	133
Правила.....	135
Умолчания.....	135
Типы специальных таблиц.....	136
Секционированные таблицы.....	137
Временные таблицы.....	137
Системные таблицы.....	137
Широкие таблицы.....	138
Вопросы производительности широких таблиц.....	138
Конструирование таблиц.....	139
Правила планирования таблиц.....	139
Присвоение типа данных столбцу.....	139
Принудительное обеспечение целостности данных.....	140
Данные в строке.....	140
Превышающие размер страницы данные строки, превышающие 8 КБ.....	144
Автоматическая нумерация и столбцы идентификаторов.....	146
Вычисляемые столбцы.....	146
Принудительное обеспечение целостности данных.....	148

Ограничения PRIMARY KEY.....	148
Ограничения FOREIGN KEY.....	149
Ограничения UNIQUE.....	156
Ограничения CHECK.....	156
Определения DEFAULT.....	157
Разрешение значений NULL.....	158
Создание и изменение таблиц.....	159
Основы создания и изменение таблиц.....	159
Свойства таблицы.....	159
Временные таблицы.....	160
Изменение таблиц.....	160
Изменение свойств столбцов.....	161
Тип данных столбца.....	161
Длина данных столбца.....	161
Точность столбца.....	161
Масштаб столбца.....	162
Возможность наличия в столбце значений NULL.....	162
Разреженные столбцы и наборы столбцов.....	162
Добавление и удаление столбцов.....	163
Использование разреженных столбцов.....	163
Свойства разреженных столбцов.....	164
Предполагаемая экономия места по типам данных.....	165
Издержки в памяти при выполнении операций обновления разреженных столбцов.....	166
Ограничения на использование разреженных столбцов.....	167
Технологии SQL Server, поддерживающие разреженные столбцы.....	168
Примеры.....	168
Использование наборов столбцов.....	170
Рекомендации по использованию наборов столбцов.....	170
Рекомендации по выбору данных из набора столбцов.....	171
Вставка или изменение данных в наборе столбцов.....	171
Использование типа данных sql_variant.....	172
Безопасность.....	173
Примеры.....	174
Создание и изменение ограничений PRIMARY KEY.....	176
Создание и изменение ограничений FOREIGN KEY.....	177
Принудительное применение ограничения FOREIGN KEY при помощи параметра WITH NOCHECK.....	178
Отключение ограничения FOREIGN KEY.....	178
Создание и изменение ограничений UNIQUE.....	179
Создание и изменение ограничений CHECK.....	180
Форсирование ограничения CHECK с помощью параметра WITH NOCHECK.....	181
Отключение ограничений CHECK.....	182
Изменение и создание определений DEFAULT.....	182
Создание и изменение столбцов идентификаторов.....	183
Свойство IDENTITY.....	184
Глобальные уникальные идентификаторы.....	184
Создание сжатых таблиц и индексов.....	186
Замечания по использованию сжатия строк и страниц.....	186
Реализация сжатия.....	188
Ожидаемая экономия при сжатии.....	188
Влияние сжатия на секционированные таблицы и индексы.....	188
Влияние сжатия на репликацию.....	189
Влияние сжатия на другие компоненты SQL Server.....	190

Запуск мастера сжатия данных.....	191
Наблюдение за сжатием.....	191
Примеры.....	191
Реализация сжатия строк.....	195
Реализация сжатия страниц.....	200
Когда происходит сжатие страницы.....	202
Реализация сжатия Юникода.....	202
Просмотр таблиц.....	204
sp_help (Transact-SQL).....	204
Синтаксис.....	204
Аргументы.....	204
Значения кодов возврата.....	204
Результирующие наборы.....	204
Замечания.....	209
Разрешения.....	209
Примеры.....	209
SELECT (Transact-SQL).....	210
Синтаксис.....	210
Замечания.....	211
Логический порядок обработки инструкции SELECT.....	211
Разрешения.....	212
Выражение SELECT (Transact-SQL).....	212
Примеры использования инструкции SELECT (Transact-SQL).....	216
COMPUTE (Transact-SQL).....	229
Предложение FOR (Transact-SQL).....	232
GROUP BY (Transact-SQL).....	237
HAVING (Transact-SQL).....	246
Предложение INTO (Transact-SQL).....	247
Предложение ORDER BY (Transact-SQL).....	251
Предложение OVER (Transact-SQL).....	253
sys.tables (Transact-SQL).....	257
Разрешения.....	260
sys.columns (Transact-SQL).....	260
Разрешения.....	262
COLUMNPROPERTY (Transact-SQL).....	262
Синтаксис.....	262
Аргументы.....	262
Типы возвращаемых данных.....	266
Исключения.....	266
Замечания.....	266
Примеры.....	266
sys.computed_columns (Transact-SQL).....	267
Разрешения.....	268
Представление каталога sys.sql_expression_dependencies (Transact-SQL).....	268
Замечания.....	272
Разрешения.....	273
Примеры.....	273
Функция динамического управления sys.dm_sql_referenced_entities (Transact-SQL)	
.....	275
Синтаксис.....	275
Аргументы.....	275
Возвращенная таблица.....	276
Исключения.....	279
Замечания.....	279

---

---

Разрешения.....	280
Примеры.....	281
sys.dm_sql_referencing_entities (Transact-SQL).....	283
Синтаксис.....	283
Аргументы.....	283
Возвращенная таблица.....	284
Исключения.....	285
Замечания.....	285
Разрешения.....	286
Примеры.....	286
Удаление таблицы.....	287
DROP TABLE (Transact-SQL).....	287
Синтаксис.....	288
Аргументы.....	288
Замечания.....	288
Разрешения.....	289
Примеры.....	289
TRUNCATE TABLE (Transact-SQL).....	290
Синтаксис.....	290
Аргументы.....	290
Замечания.....	290
Ограничения.....	291
Усечение больших таблиц.....	291
Разрешения.....	291
Примеры.....	292
Удаление и повторная сборка больших объектов.....	292



# Архитектура страниц и экстенгов<sup>i</sup>

Страница — основная единица хранения данных в SQL Server. Экстент — это набор из 8 физически непрерывных страниц. Экстенги помогают эффективно управлять страницами.

В этом разделе описаны структуры данных, используемые для управления страницами и экстенгами. Понимание архитектуры страниц и экстенгов важно для проектирования и разработки эффективных баз данных.

## Страницы и экстенги

Основной единицей хранилища данных в SQL Server является страница. Место на диске, предоставляемое для размещения файла данных (MDF- или NDF-файл) в базе данных, логически разделяется на страницы с непрерывным перечислением от 0 до n. Дисковые операции ввода-вывода выполняются на уровне страницы. А именно, SQL Server считывает или записывает целые страницы данных.

Экстент — это коллекция, состоящая из восьми физически непрерывных страниц; они используются для эффективного управления страницами. Все страницы хранятся в экстенгах.

## Страницы

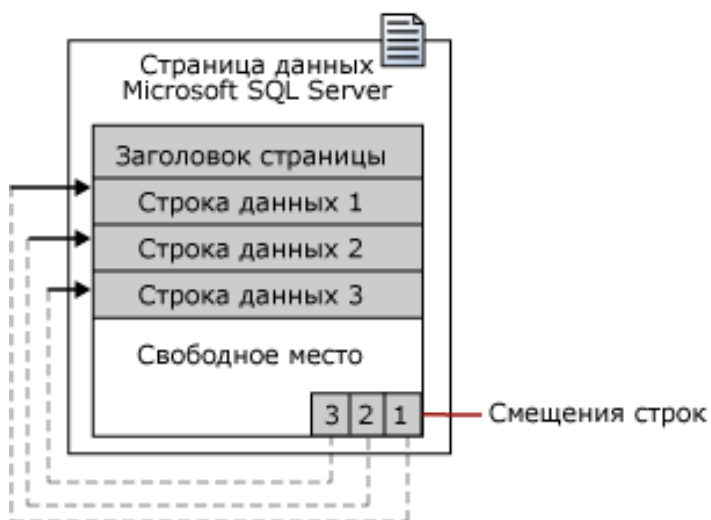
В SQL Server размер страницы составляет 8 КБ. Это значит, что в одном мегабайте базы данных SQL Server содержится 128 страниц. Каждая страница начинается с 96-байтового заголовка, который используется для хранения системных данных о странице. Эти данные включают номер страницы, тип страницы, объем свободного места на странице и идентификатор единицы распределения объекта, которому принадлежит страница.

В следующей таблице представлены типы страниц, используемые в файлах данных базы данных SQL Server.

Тип страницы	Содержимое
Данные	Строки данных со всеми данными, кроме данных типа text, ntext, image, nvarchar(max), varchar(max) и varbinary(max), а также данными типа xml, когда параметр <b>текст в строке</b> установлен в значение ON.
Индекс	Записи индекса.
Текст/изображение	Типы данных больших объектов: <ul style="list-style-type: none"><li>• text, ntext, image, nvarchar(max), varchar(max), varbinary(max) и xml.</li></ul>

	Столбцы переменной длины, когда строки данных превышают размер 8 КБ: <ul style="list-style-type: none"> <li>• <code>varchar</code>, <code>nvarchar</code>, <code>varbinary</code> и <code>sql_variant</code>.</li> </ul>
Глобальная карта распределения, общая глобальная карта распределения	Сведения о том, размещены ли экстенги.
Свободное место на страницах	Сведения о размещении страниц и доступном на них свободном месте.
Карта распределения индекса	Сведения об экстенгах, используемых таблицей или индексом для единицы распределения.
Схема массовых изменений	Сведения об экстенгах, измененных массовыми операциями со времени последнего выполнения инструкции <code>BACKUP LOG</code> для единицы распределения.
Схема разностных изменений	Сведения об экстенгах, измененных с момента последнего выполнения инструкции <code>BACKUP DATABASE</code> для единицы распределения.
<b>Примечание</b>	
Файлы журнала не содержат страниц, в них содержится последовательность записей журнала.	

Строки данных заносятся на страницу последовательно, сразу же после заголовка. Таблица смещения строк начинается в конце страницы; каждая таблица смещения строк содержит одну запись для каждой строки на странице. Каждая запись регистрирует, насколько далеко от начала страницы находится первый байт строки. Записи в таблице смещения строк находятся в обратном порядке относительно последовательности строк на странице.



### Поддержка больших строк

Строка не может разделить страницу на части, однако часть строки может быть перемещена на другую страницу, чтобы строка действительно была очень большой. Максимальный объем данных и служебного кода, содержащихся в одной строке на странице, составляет 8 060 байт (8 КБ). Однако сюда не включены данные, хранимые в типе страницы «Текст/изображение». Это ограничение не такое строгое для таблиц, содержащих столбцы, относящиеся к типам данных `varchar`, `nvarchar`, `varbinary` или `sql_variant`. Когда общий размер строк всех фиксированных и переменных столбцов в таблице превышает предел в 8 060 байт, SQL Server динамически перемещает один или более столбцов переменной длины на страницы в единице распределения `ROW_OVERFLOW_DATA`, начиная со столбца с наибольшей шириной. Это действие выполняется всегда, когда в результате операций вставки или обновления общий размер строки выходит за предел в 8 060 байт. Когда происходит перемещение столбца на страницу в единице распределения `ROW_OVERFLOW_DATA`, 24-байтовый указатель на исходной странице в единице распределения `IN_ROW_DATA` сохраняется. Если при последующей выполняемой операции размер строки уменьшается, SQL Server динамически перемещает столбцы обратно на исходную страницу данных. Дополнительные сведения см. в разделе [Превышающие размер страницы данные строки, превышающие 8 КБ](#).

## Экстенги

Экстенги являются основными единицами организации пространства. Экстент состоит из восьми непрерывных страниц или 64 КБ. Это значит, что в одном мегабайте базы данных SQL Server содержится 16 экстенгов.

Чтобы сделать распределение места эффективным, SQL Server не размещает целые экстенги в таблицы с небольшим объемом данных. SQL Server имеет два типа экстенгов.

- Однородные экстенги принадлежат одному объекту; все восемь страниц в кластере могут быть использованы только этим владеющим объектом.
- Смешанные экстенги могут находиться в общем пользовании у не более восьми объектов. Каждая из восьми страниц в экстенге может находиться во владении

разных объектов.

Новая таблица или индекс — это обычно страницы, выделенные из смешанных экстентов. При увеличении размера таблицы или индекса до восьми страниц эти таблица или индекс переходят на использование однородных экстентов для последовательных единиц распределения. При создании индекса для существующей таблицы, в которой содержится достаточно строк, чтобы сформировать восемь страниц в индексе, все единицы распределения для индекса находятся в однородных экстентах.



## Управление размещением экстента и свободным местом

Структуры данных SQL Server, управляющие размещением экстента и отслеживанием свободного места, обладают относительно простой структурой. Это позволяет получить следующие преимущества:

- Сведения о свободном месте плотно упакованы, поэтому эти данные содержат относительно небольшое количество страниц. Это приводит к увеличению скорости из-за уменьшения необходимых операций чтения диска для получения сведений о размещении. Также увеличивается вероятность того, что страницы размещения будут оставаться в памяти и повторных операций чтения не потребуется.
- Большая часть сведений о размещении не связана по цепочке друг с другом. Это упрощает управление сведениями о размещении. Каждое действие по размещению или освобождению страницы может выполняться быстро. Это сокращает конфликты между одновременными задачами размещения и освобождения страниц.

## Управление размещением экстента

SQL Server использует два типа карт размещения для записи размещения экстентов:

- Глобальная карта распределения (GAM)  
На GAM-страницах записано, какие экстен-ты были размещены. В каждой карте GAM содержится 64 000 экстен-тов или почти 4 ГБ данных. В карте GAM приходится по одному биту на каждый экстен-т в покрываемом им интервале. Если бит равен 1, то экстен-т свободен; если бит равен 0, то экстен-т размещен.
- Общая глобальная карта распределения (SGAM)  
На SGAM-страницах записано, какие экстен-ты в текущий момент используются в качестве смешанных экстен-тов и имеют как минимум одну неиспользуемую страницу. В каждой карте SGAM содержится 64 000 экстен-тов или почти 4 ГБ данных. В карте SGAM приходится по одному биту на каждый экстен-т в покрываемом им интервале. Если бит равен 1, то экстен-т используется как

смешанный экстент и имеет свободную страницу. Если бит равен 0, то экстент не используется как смешанный экстент, или он является смешанным экстендом, но все его страницы используются.

Каждый экстент обладает следующими наборами битовых шаблонов в картах GAM и SGAM, основанными на его текущем использовании.

Текущее использование экстента	Настройка битов карты GAM	Настройка битов карты SGAM
Свободно, в текущий момент не используется	1	0
Однородный экстент или заполненный смешанный экстент	0	0
Смешанный экстент со свободными страницами	0	1

Это дает простые алгоритмы управления экстендами страниц. Для размещения однородного экстента компонент Database Engine производит на карте GAM поиск бита 1 и заменяет его на бит 0. Для поиска смешанного экстента со свободными страницами компонент Database Engine производит поиск на карте SGAM бита 1. Для размещения смешанного экстента компонент Database Engine производит на карте GAM поиск бита 1 и заменяет его на бит 0, а затем устанавливает значение соответствующего бита на карте SGAM равным 1. Для освобождения экстента компонент Database Engine устанавливает бит GAM равным 1, а соответствующий бит SGAM равным 0. Внутренние алгоритмы, которые на самом деле используются компонентом Database Engine, более сложны, чем это описано в данном подразделе, так как компонент Database Engine распространяет данные в базе данных равномерно. Однако даже настоящие алгоритмы упрощаются из-за того, что отпадает необходимость управления цепочками сведений о размещении экстенгов.

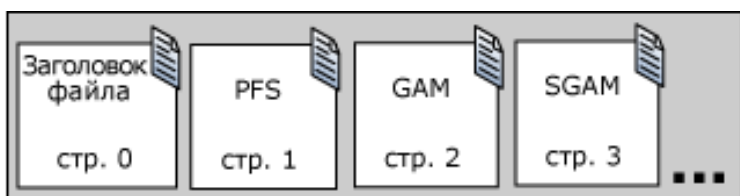
## Отслеживание свободного места

На страницы PFS (Page Free Space) записывается состояние размещения каждой страницы, информация о том, была ли отдельная страница размещена или нет, а также количество свободного места на каждой странице. В PFS на каждую страницу приходится по одному байту, хранящему информацию о том, была ли страница размещена или нет, а если была — то пустая она, или ее заполнение находится в промежутке от 1 до 50 процентов, от 51 до 80 процентов, от 81 до 95 процентов или от 96 до 100 процентов.

После размещения экстента на объект компонент Database Engine использует PFS-страницы для записи информации о том, какие страницы в экстенде размещены, а какие свободны. Эти сведения используются компонентом Database Engine при размещении новой страницы. Количеством свободного места на странице можно управлять только в случае кучи и страниц Text/Image. Это используется при поиске страницы компонентом

Database Engine, обладающей свободным местом, достаточным для сохранения в ней новой добавляемой строки. Для индексов не требуется, чтобы отслеживалось свободное место на странице, так как место, в которое будет вставляться новая строка, назначается значениями ключа индекса.

PFS-страница является первой страницей после страницы заголовка файла в файле данных (страница номер 1). Потом следует GAM-страница (страница номер 2), а затем SGAM-страница (страница номер 3). После первой PFS-страницы находится PFS-страница размером примерно 8 000 страниц. После первой GAM-страницы на странице 2 находится другая GAM-страница с 64 000 экстентов и другая SGAM-страница с 64 000 экстентов находится после первой SGAM-страницы на странице 3. На следующей иллюстрации показана последовательность страниц, используемая компонентом Database Engine, для размещения и управления экстентами.



## Управление дисковым пространством, занятым объектами

Страница карты распределения индекса (Index Allocation Map, IAM) сопоставляет экстенты в 4-гигабайтном фрагменте файла базы данных с используемой единицей распределения. Единица распределения может иметь один из трех типов.

- IN\_ROW\_DATA

Содержит секцию кучи или индекса.

- LOB\_DATA

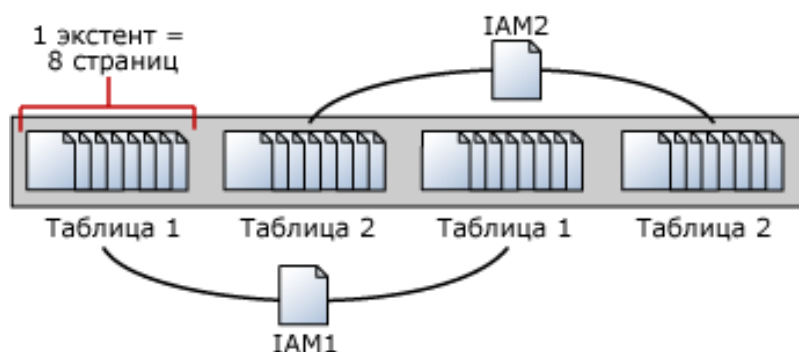
Содержит типы данных больших объектов (LOB), например: xml, varbinary(max) или varchar(max).

- ROW\_OVERFLOW\_DATA

Содержит данные переменной длины, которые хранятся в столбцах varchar, nvarchar, varbinary или sql\_variant и превышают допустимый размер строки 8060 байт.

В каждой секции кучи или индекса содержится по крайней мере одна единица распределения IN\_ROW\_DATA. Кроме того, в зависимости от схемы кучи или индекса, там могут содержаться единицы распределения LOB\_DATA или ROW\_OVERFLOW\_DATA. Дополнительные сведения о единицах распределения см. в разделе [Организация таблиц и индексов](#).

IAM-страница охватывает в файле диапазон 4 ГБ, то есть столько же, сколько и GAM-или SGAM-страница. Если в единице распределения содержатся экстенты из более чем одного файла или фрагмент файла размером более 4 ГБ, то несколько IAM-страниц будут объединены в IAM-цепочку. Таким образом, каждая единица распределения содержит как минимум одну IAM-страницу для каждого из файлов, в которых содержатся ее экстенты. Для файла может существовать несколько IAM-страниц, если размер экстентов файла, назначенного единице распределения, превышает объем, который может быть записан в одной IAM-странице.

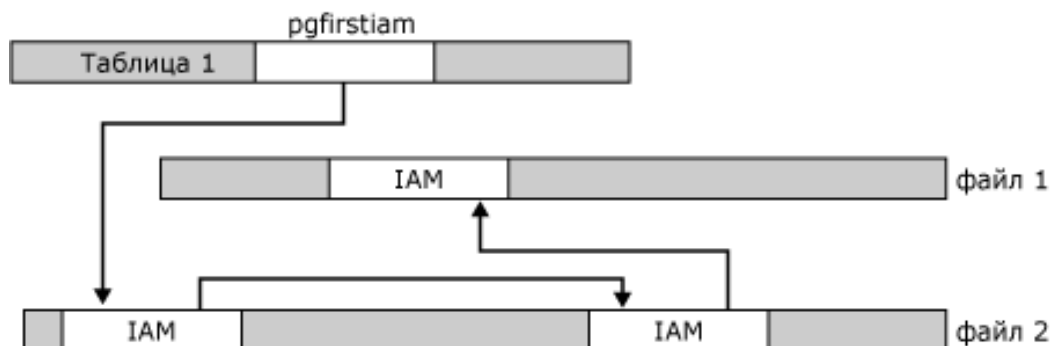


IAM-страницы для каждой единицы распределения выделяются по необходимости и располагаются в файле в случайном порядке. Системное представление **sys.system\_internals\_allocation\_units** указывает на первую IAM-страницу единицы распределения. Все IAM-страницы, относящиеся к одной единице распределения, объединяются в цепочку.

### Важно!

Системное представление **sys.system\_internals\_allocation\_units** предназначено только для внутреннего использования и может быть изменено. Совместимость не гарантируется.

**sys.system\_internals\_allocation\_units**



AM-страница имеет заголовок, отражающий первый экстент из диапазона, сопоставленного с данной IAM-страницей. IAM-страница также имеет большую битовую карту, в которой каждый бит представляет экстент. Первый бит схемы представляет первый экстент диапазона, второй бит — второй экстент и т. д. Если бит равен 0, то соответствующий ему экстент не привязан к единице распределения, которой принадлежит IAM-страница. Если он равен 1, то соответствующий ему экстент привязан к единице распределения, которой принадлежит IAM-страница.

Когда требуется вставить новую строку, но на текущей странице нет свободного места, компонент SQL Server Database Engine через IAM и PFS ищет страницу для выделения или (для страниц кучи, или текста и изображения) страницу, в которой достаточно места для хранения строки данных. Используя IAM-страницы, компонент Database Engine находит экстенды, привязанные к единице распределения. Для каждого экстенды компонент Database Engine просматривает PFS-страницы, чтобы определить наличие доступных страниц. Огромное число страниц данных содержатся в сравнительно небольшом числе IAM- и PFS-страниц. Поэтому обычно IAM- и PFS-страницы находятся в

памяти буферного пула SQL Server, и поиск в них осуществляется очень быстро. Для индексов место вставки новой строки определяется ключом индекса. В этом случае описанный ранее процесс поиска не производится.

Компонент Database Engine размещает новый экстент для единицы распределения только в том случае, когда в существующем экстенте невозможно быстро найти страницу, в которой достаточно места для вставляемой строки. Для размещения экстентов в файловой группе компонент Database Engine пользуется пропорциональным алгоритмом размещения. Если файловая группа состоит из двух файлов и в одном из них свободного места вдвое больше, чем в другом, то во втором файле будет размещаться по одной странице на каждые две страницы во первом. Это означает, что процент заполнения для каждого из файлов в группе будет примерно одинаковым.

## Отслеживание измененных экстентов

SQL Server использует две структуры внутренних данных, чтобы отслеживать экстенты, измененные операциями массового копирования, и экстенты, измененные со времени последнего полного резервного копирования. Эти структуры данных существенно ускоряют разностные резервные копии. Они также ускоряют операции записи в журнал массового копирования, если база данных использует модель восстановления с неполным протоколированием. Подобно страницам глобальной карты распределения (GAM) и общей общей глобальной карты распределения (SGAM), эти структуры являются битовыми картами, в которых каждый бит представляет один экстент.

- **Схема разностных изменений (DCM)**

Она отслеживает экстенты, которые были изменены со времени последнего выполнения инструкции BACKUP DATABASE. Если бит или экстент равен 1, значит, экстент был изменен со времени последнего выполнения инструкции BACKUP DATABASE. Если бит равен 0, то экстент не был изменен.

Чтобы определить, какие экстенты были изменены, разностные резервные копии считывают только страницы DCM. Это существенно сокращает количество страниц, которые должна просмотреть разностная резервная копия. Количество времени, которое затрачивает разностная резервная копия, пропорционально количеству экстентов, измененных со времени последнего выполнения инструкции BACKUP DATABASE, а не размеру всей базы данных.

- **Схема массовых изменений (BCM)**

Она отслеживает экстенты, измененные операциями с неполным протоколированием со времени последнего выполнения инструкции BACKUP LOG. Если бит или кластер страниц равен 1, значит, экстент был изменен операцией неполного протоколирования после последнего выполнения инструкции BACKUP LOG. Если бит равен 0, то экстент не был изменен операциями с неполным протоколированием.

Несмотря на то, что страницы BCM существуют во всех базах данных, они применяются только в том случае, если база данных использует модель восстановления с неполным протоколированием. В данной модели восстановления при выполнении инструкции BACKUP LOG процесс резервного копирования просматривает схемы BCM для измененных экстентов. Затем он включает экстенты из резервной копии журнала. Это позволяет восстановиться операциям с неполным протоколированием, если база данных восстанавливается из резервной копии журнала и последовательности резервных копий журнала



транзакции. Страницы BCM в базе данных, которая использует простую модель восстановления, не соответствуют, потому что не произведена запись в журнал ни одной операции с неполным протоколированием. Они не соответствуют базе данных, которая использует модель полного восстановления, потому что эта модель восстановления принимает операции с неполным протоколированием за операции с полным протоколированием.

Интервал между DCM- и BCM-страницами равен интервалу между GAM- и SGAM-страницами — 64 000 экстендов. DCM- и BCM-страницы расположены за GAM- и SGAM-страницами в физическом файле:



## Структура журнала транзакций

Этот раздел содержит основные сведения об архитектуре, необходимые для понимания того, как журнал транзакций обеспечивает целостность данных в базе данных и как он используется для восстановления данных.

Сведения, изложенные в этом разделе, помогают повысить эффективность работы с журналами транзакций.

## Логическая архитектура журнала транзакций

Логически журнал транзакций SQL Server работает так, как если бы он являлся последовательностью записей в журнале. Каждая запись журнала идентифицируется регистрационным номером транзакции (номер LSN). Каждая новая запись добавляется в логический конец журнала с номером LSN, который больше номера LSN предыдущей записи.

Записи журнала хранятся в той последовательности, в которой они были созданы. Каждая запись журнала содержит идентификатор транзакции, к которой она относится. Все записи журнала, связанные с определенной транзакцией, с помощью обратных указателей связаны в цепочку, которая предназначена для ускорения отката транзакции.

Записи журнала для изменения данных содержат либо выполненную логическую операцию, либо исходный и результирующий образ измененных данных. Исходный образ записи — это копия данных до выполнения операции, а результирующий образ — копия данных после ее выполнения.

Действия, которые необходимо выполнить для восстановления операции, зависят от типа журнальной записи:

- Зарегистрирована логическая операция.
  - Для наката логической операции выполняется эта операция.
  - Для отката логической операции выполняется логическая операция, обратная зарегистрированной.

- Зарегистрированы исходный и результирующий образы записи.
- Для наката операции применяется результирующий образ.
- Для отката операции применяется исходный образ.

В журнал транзакций записываются различные типы операций, например:

- начало и конец каждой транзакции;
- любые изменения данных (вставка, обновление или удаление), включая изменения в любой таблице (в том числе и в системных таблицах), производимые системными хранимыми процедурами или инструкциями языка DDL;
- любое выделение и освобождение страниц и экстентов;
- создание и удаление таблиц и индексов.

Кроме того, регистрируются операции отката. Каждая транзакция резервирует в журнале транзакций место, чтобы при выполнении инструкции отката или возникновения ошибки в журнале было достаточно места для регистрации отката. Объем резервируемого пространства зависит от выполняемых в транзакции операций, но обычно он равен объему, необходимому для регистрации каждой из операций. Все это пространство после завершения транзакции освобождается.

Раздел журнального файла, который начинается от первой записи, необходимой для успешного отката на уровне базы данных, до последней зарегистрированной записи называется активной частью журнала, или активным журналом. Именно этот раздел необходим для выполнения полного восстановления базы данных. Активный журнал не может быть усечен.

## Физическая архитектура журнала транзакций

Журнал транзакций используется для обеспечения целостности данных в базе данных и восстановления данных. В этом подразделе приводятся сведения о физической структуре журнала транзакций. Понимание физической архитектуры может помочь в повышении эффективности работы с журналами транзакций.

Журнал транзакций в базе данных сопоставляет один или несколько физических файлов. По сути, файл журнала представляет собой строку записей журнала. Физически последовательность записей журнала эффективно хранится в наборе физических файлов, которые образуют журнал транзакций.

Компонент SQL Server Database Engine делит каждый физический файл журнала на несколько виртуальных файлов журнала. Виртуальные файлы журнала не имеют фиксированных размеров. Не существует также и определенного числа виртуальных файлов журнала, приходящихся на один физический файл журнала. Компонент Database Engine динамически определяет размер виртуальных файлов журнала при создании или расширении файлов журнала. Компонент Database Engine стремится обслуживать небольшое число виртуальных файлов. После расширения файла журнала размер виртуальных файлов определяется как сумма размера существующего журнала и размера нового приращения файла. Администраторы не могут настраивать или устанавливать размеры и число виртуальных файлов журнала.

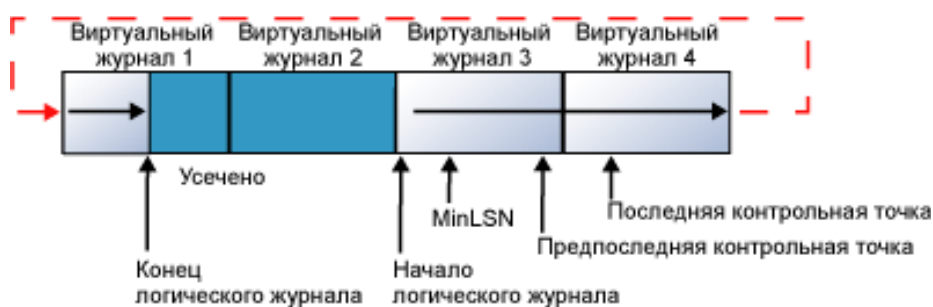
Виртуальные файлы журнала влияют на производительность системы лишь в том случае, если эти файлы журнала определяются малыми значениями `size` и `growth_increment`.

Если эти файлы журнала в результате большого числа малых приращений будут разрастаться до крупных размеров, они будут иметь множество виртуальных файлов журнала. В итоге может увеличиться время запуска базы данных, а также снизиться скорость операций резервного копирования и восстановления. Рекомендуется выбирать для файлов журнала значение параметра `size`, близкое к окончательному требуемому размеру, а также задавать относительно большое значение `growth_increment`.

Журнал транзакций является оборачиваемым файлом. Рассмотрим пример. Пусть база данных имеет один физический файл журнала, разделенный на четыре виртуальных файла журнала. При создании базы данных логический файл журнала начинается в начале физического файла журнала. Новые записи журнала добавляются в конце логического журнала и приближаются к концу физического файла журнала. Усечение журнала освобождает любые виртуальные журналы, все записи которых находятся перед минимальным регистрационным номером восстановления в журнале транзакций (MinLSN). MinLSN является в журнале транзакций регистрационным номером самой старой записи, которая необходима для успешного отката на уровне всей базы данных. Журнал транзакций рассматриваемой в данном примере базы данных будет выглядеть примерно так же, как на следующей иллюстрации.



Когда конец логического журнала достигнет конца физического файла журнала, новые записи журнала будут размещаться в начале физического файла журнала.



Этот цикл повторяется бесконечно, пока конец логического журнала не совмещается с началом этого логического журнала. Если старые записи журнала усекаются достаточно часто, так что при этом всегда остается место для новых записей журнала, созданных с новой контрольной точки, журнал постоянно остается незаполненным. Однако, если конец логического журнала совмещается с началом этого логического журнала, происходит одно из двух событий, перечисленных ниже.

- Если для данного журнала применена установка `FILEGROWTH` и на диске имеется свободное место, файл расширяется на величину, указанную в `growth_increment`, и новые записи журнала добавляются к этому расширению.

Дополнительные сведения о настройке FILEGROWTH см. в разделе [ALTER DATABASE \(Transact-SQL\)](#).

- Если установка FILEGROWTH не применяется или диск, на котором размещается файл журнала, имеет меньше свободного места, чем это указано в *growth\_increment*, формируется ошибка 9002.

Если в журнале содержится несколько физических файлов журнала, логический журнал будет продвигаться по всем физическим файлам журнала до тех пор, пока он не вернется на начало первого физического файла журнала.

## Контрольные точки и активная часть журнала

При достижении контрольных точек измененные страницы данных записываются из буферного кэша текущей базы данных на диск. Это сводит к минимуму активную часть журнала, которую приходится обрабатывать при полном восстановлении базы данных. Во время полного восстановления базы данных выполняются следующие действия.

- Накат записанных в журнал изменений, не записанных на диск до остановки системы.
- Откат всех изменений, связанных с незавершенными транзакциями, такими как транзакции, для которых в журнале нет записи COMMIT или ROLLBACK.

## Функционирование контрольной точки

Контрольная точка выполняет в базе данных следующее.

- Записывает в файл журнала запись, отмечающую начало контрольной точки.
- Сохраняет данные, записанные для контрольной точки в цепи записей журнала контрольной точки.

Одним из элементов данных, регистрируемых в записях контрольной точки, является номер LSN первой записи журнала, при отсутствии которой успешный откат в масштабе всей базы данных невозможен. Такой номер LSN называется минимальным номером LSN восстановления (MinLSN). Номер MinLSN является наименьшим значением из:

- номера LSN начала контрольной точки;
- номера LSN начала старейшей активной транзакции;
- номера LSN начала старейшей транзакции репликации, которая еще не была доставлена базе данных распространителя.

Записи контрольной точки содержат также список активных транзакций, изменивших базу данных.

- Если база данных использует простую модель восстановления, помечает для повторного использования пространство, предшествующее номеру MinLSN.
- Записывает все измененные страницы журналов и данных на диск.
- Записывает в файл журнала запись, отмечающую конец контрольной точки.
- Записывает в страницу загрузки базы данных номер LSN начала соответствующей цепи.

## Действия, приводящие к срабатыванию контрольных точек

Контрольные точки срабатывают в следующих ситуациях.

- При явном выполнении инструкции CHECKPOINT. Контрольная точка

- срабатывает в текущей базе данных соединения.
- При выполнении в базе данных операции с минимальной регистрацией, например при выполнении операции массового копирования для базы данных, на которую распространяется модель восстановления с неполным протоколированием.
- При добавлении или удалении файлов баз данных с использованием инструкции ALTER DATABASE.
- При остановке экземпляра SQL Server с помощью инструкции SHUTDOWN или при остановке службы SQL Server (MSSQLSERVER). И в том, и в другом случае будет создана контрольная точка каждой базы данных в экземпляре SQL Server.
- Если экземпляр SQL Server периодически создает в каждой базе данных автоматические контрольные точки для сокращения времени восстановления базы данных.
- При создании резервной копии базы данных.
- При выполнении действия, требующего отключения базы данных. Примерами могут служить присвоение параметру AUTO\_CLOSE значения ON и закрытие последнего соединения пользователя с базой данных или изменение параметра базы данных, требующее перезапуска базы данных.

## Автоматические контрольные точки

Компонент SQL Server Database Engine создает контрольные точки автоматически. Интервал между автоматическими контрольными точками определяется на основе использованного места в журнале и времени, прошедшего с момента создания последней контрольной точки. Интервал между автоматическими контрольными точками колеблется в широких пределах и может быть довольно длительным, если база данных изменяется редко. При крупномасштабных изменениях данных частота автоматических контрольных точек может быть гораздо выше.

Используйте параметр конфигурации сервера **recovery interval** для вычисления интервала между автоматическими контрольными точками для всех баз данных на экземпляре сервера. Значение этого параметра определяет максимальное время, отводимое компоненту Database Engine на восстановление базы данных при перезапуске системы. Компонент Database Engine оценивает количество записей журнала, которые он может обработать за время **recovery interval** при выполнении операции восстановления.

Кроме того, интервал между автоматическими контрольными точками зависит от модели восстановления, как показано ниже.

- Если применяется полная модель восстановления или модель восстановления с неполным протоколированием, то автоматическая контрольная точка создается каждый раз, когда число записей в журнале достигает значения, определенного Database Engine в качестве предельного количества записей, которое оно может обработать за время, заданное параметром **recovery interval**.
- Если используется простая модель восстановления базы данных, автоматическая контрольная точка создается каждый раз, когда число записей в журнале достигает меньшего из двух предельных условий:
  - журнал заполняется на 70 процентов;
  - число записей в журнале достигает значения, определенного компонентом Database Engine в качестве количества записей, которое он может обработать за время, заданное параметром **recovery interval**.

Сведения об установке интервала восстановления см. в разделе [Как установить интервал восстановления \(среда SQL Server Management Studio\)](#).

#### Совет

Дополнительный параметр -k программы установки SQL Server позволяет администратору базы данных регулировать контрольные точки ввода/вывода на основании пропускной способности подсистемы ввода/вывода для некоторых типов контрольных точек. Параметр -k программы установки применяется к автоматическим контрольным точкам и любым другим контрольным точкам без дросселирования.

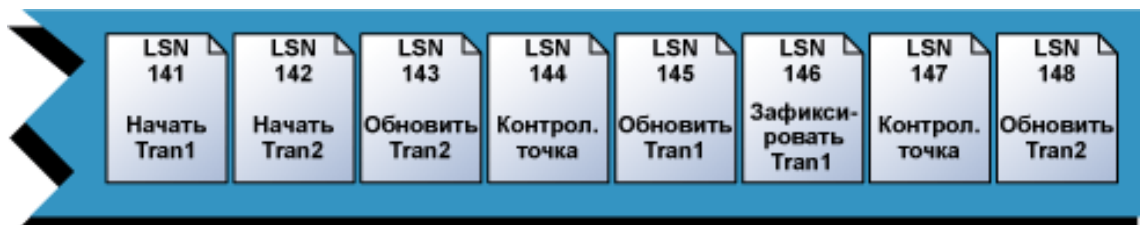
Если используется простая модель восстановления базы данных, то при срабатывании автоматических контрольных точек неиспользуемая часть журнала транзакций удаляется. Однако при использовании модели полного восстановления или модели восстановления с неполным протоколированием журнал в результате срабатывания автоматических контрольных точек не усекается. Дополнительные сведения см. в разделе [Усечение журнала транзакций](#).

Инструкция CHECKPOINT теперь поддерживает необязательный аргумент checkpoint\_duration, определяющий время (в секундах), отводимое контрольным точкам на завершение. Дополнительные сведения см. в разделе [CHECKPOINT \(Transact-SQL\)](#).

## Активный журнал

Часть журнала, начинающаяся с номера MinLSN и заканчивающаяся последней записью, называется активной частью журнала, или активным журналом. Этот раздел журнала необходим для выполнения полного восстановления базы данных. Ни одна часть активного журнала не может быть усечена. Все записи журнала до номера MinLSN должны быть удалены из частей журнала.

На следующем рисунке изображена упрощенная схема журнала завершения транзакций, содержащего две активные транзакции. Записи контрольных точек были сжаты в одну запись.



Последней записью в журнале транзакций является запись с номером LSN, равным 148. На момент обработки записанной контрольной точки с номером LSN 147 транзакция 1 уже зафиксирована и единственной активной транзакцией является транзакция 2. В результате первая запись журнала, созданная для транзакции 2, становится старейшей записью активной транзакции на момент последней контрольной точки. Таким образом, номером MinLSN становится номер LSN, равный 142 и соответствующий записи начала транзакции 2.

## Длительные транзакции

Активный журнал должен включать в себя все элементы всех незафиксированных транзакций. Приложение, инициирующее транзакцию и не выполняющее ее фиксацию или откат, не позволяет компоненту Database Engine повышать MinLSN. Это может привести к проблемам двух типов.

- Если система будет выключена после того, как транзакцией было выполнено много незафиксированных изменений, этап восстановления при последующем перезапуске может занять гораздо больше времени, чем указано параметром **recovery interval**.
- Журнал может достичь очень большого объема, потому что после номера MinLSN усечь его нельзя. Это справедливо даже в том случае, если используется простая модель восстановления, когда журнал транзакций обычно усекается при каждой автоматической контрольной точке.

## Транзакции репликации

Агент чтения журнала следит за журналом транзакций всех баз данных, на которых настроена репликация транзакций, и копирует отмеченные для репликации транзакции из журнала транзакций в базу данных распространителя. Активный журнал должен содержать все транзакции, отмеченные для репликации, но еще не доставленные базе данных распространителя. Если эти транзакции не реплицировать в допустимый срок, усечение журнала может оказаться невозможным. Дополнительные сведения см. в разделе [Как работает репликация транзакций](#).

## Журнал транзакций с упреждающей записью

В этом разделе описана роль, которую журнал транзакций с упреждающей записью играет в записи изменений данных на диск. Основные сведения о журналах транзакций см. в разделе [Общие сведения о журналах транзакций](#).

SQL Server использует журнал с упреждающей записью, который гарантирует, что до занесения на диск записи, связанной с журналом, никакие изменения данных записаны не будут. Таким образом обеспечиваются свойства ACID для транзакции.

Дополнительные сведения о транзакциях и свойствах ACID см. в разделе [Транзакции \(компонент Database Engine\)](#).

Для понимания принципов работы журнала с упреждающей записью важно знать принципы записи измененных данных на диск. SQL Server поддерживает буферный кэш, из которого система считывает страницы данных при извлечении необходимых данных. Изменения данных не заносятся непосредственно на диск, а записываются на копии страницы в буферном кэше. Изменение не записывается на диск, пока в базе данных не возникает контрольная точка, или же изменение должно быть записано на диск таким образом, чтобы для хранения новой страницы мог использоваться буфер. Запись измененной страницы данных из буферного кэша на диск называется сбросом страницы на диск. Страница, измененная в кэше, но еще не записанная на диск, называется грязной страницей.

Во время изменения страницы в буфере запись журнала строится в кэше журнала, который записывает изменение. Данная запись журнала должна быть перенесена на диск до того, как соответствующая «грязная» страница будет записана из буферного кэша на диск. Если «грязная» страница переносится на диск до записи журнала, эта страница создает изменение на диске, которое не может быть отквачено, если сервер выйдет из строя до переноса записи журнала на диск. SQL Server обладает алгоритмом, который защищает «грязную» страницу от записи на диск до переноса на него соответствующей записи журнала. Содержимое журнала запишется на диск после того, как будут зафиксированы транзакции.

## Архитектура таблиц и структур данных индекса

В базах данных SQL Server объекты хранятся в виде коллекций страниц объемом по 8 КБ. В данном разделе приводится описание того, как организованы страницы таблиц и индексов, как они хранятся и как осуществляется доступ к этим страницам.

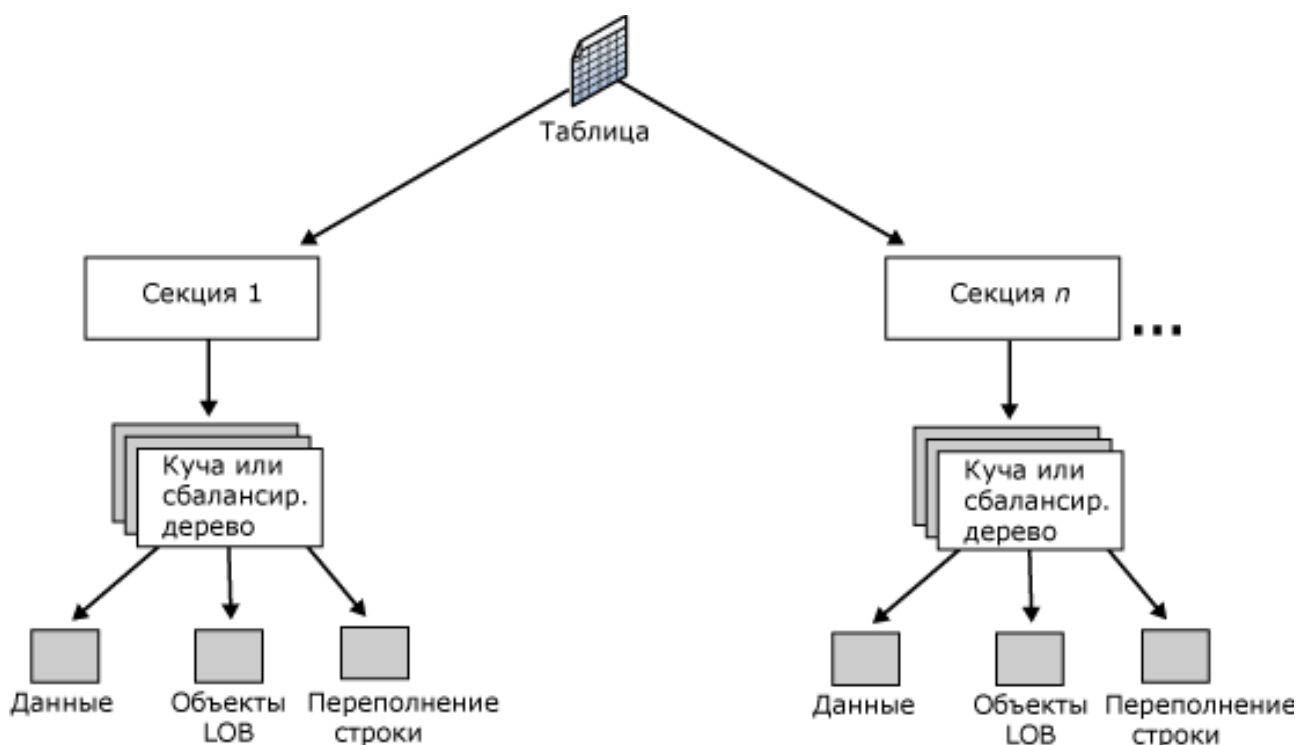
### Организация таблиц и индексов

Таблицы и индексы хранятся в виде коллекции страниц размером 8 КБ. В этом подразделе описывается способ организации страниц таблиц и индексов.

### Организация таблиц

Следующая иллюстрация показывает организацию таблицы. Таблица содержится в одной или нескольких секциях, а каждая секция содержит строки данных либо в куче, либо в структуре кластеризованного индекса. Страницы кучи или кластеризованного индекса объединяются в одну или несколько единиц распределения в зависимости от типов столбцов в строках данных.





## Секции

Страницы таблиц и индексов содержатся в одной или нескольких секциях. Секция — это пользовательская единица организации данных. По умолчанию таблица или индекс имеет единственную секцию, которая содержит все страницы таблицы или индекса. Секция располагается в одной файловой группе. Таблица или индекс, имеющие одну секцию, эквивалентны организационной структуре таблиц и индексов предыдущих версий SQL Server.

Если таблица или индекс используют несколько секций, данные разделяются горизонтально, так что группы строк сопоставляются отдельным секциям, основываясь на указанном столбце. Секции могут храниться в одной или нескольких файловых группах в базе данных. Таблица или индекс рассматриваются как единая логическая сущность при выполнении над данными запросов или обновлений. Дополнительные сведения см. в разделе [Секционированные таблицы и индексы](#).

Для просмотра секций, используемых таблицей или индексом, можно воспользоваться представлением каталога [sys.partitions \(Transact-SQL\)](#).

## Кластеризованные таблицы, кучи и индексы

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

- Кластеризованные таблицы — это таблицы, имеющие кластеризованный индекс. Строки данных хранятся по порядку ключа кластеризованного индекса. Кластеризованный индекс реализуется в виде структуры индекса сбалансированного дерева, которая поддерживает быстрый поиск строк по их ключевым значениям в кластеризованном индексе. Страницы в каждом уровне

индекса, включая страницы данных на конечном уровне, связаны в двунаправленный список. Однако перемещение из одного уровня на другой выполняется при помощи ключевых значений. Дополнительные сведения см. в разделе [Структуры кластеризованного индекса](#).

- Кучи — это таблицы, которые не имеют кластеризованного индекса. Строки данных хранятся без определенного порядка, и какой-либо порядок в последовательности страниц данных отсутствует. Страницы данных не связаны в связный список. Дополнительные сведения см. в разделе [Структуры кучи](#).

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

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

### Некластеризованные индексы

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

Дополнительные сведения см. в разделе [Структуры некластеризованных индексов](#).

### XML-индексы

Для каждого столбца xml в таблице могут быть созданы один первичный и несколько вторичных XML-индексов. XML-индекс представляет собой разделенное и сохраненное представление больших двоичных объектов XML (BLOB) в столбце типа данных xml. XML-индексы хранятся во внутренних таблицах. Для просмотра сведений об XML-индексах можно воспользоваться представлениями каталогов [sys.xml\\_indexes](#) и [sys.internal\\_tables](#).

Дополнительные сведения об XML-индексах см. в разделе [Индексы для столбцов типа данных xml](#).

## Единицы распределения

Единица распределения — это коллекция страниц в куче или сбалансированном дереве, используемая для управления данными на основании типов страниц. В следующей таблице перечисляются единицы распределения, используемые для управления данными в таблицах и индексах.

Тип единицы распределения	Данные, для управления которыми этот тип используется
---------------------------	---

IN_ROW_DATA	Строки данных или индекса, которые содержат все данные, кроме данных больших объектов (LOB). Страницы имеют тип Data или Index.
LOB_DATA	Данные большого объекта, хранимые в одном или нескольких из следующих типов данных: text, ntext, image, xml, varchar(max), nvarchar(max), varbinary(max) или определяемые пользователем типы CLR (CLR UDT). Страницы имеют тип Text/Image.
ROW_OVERFLOW_DATA	Данные переменной длины, хранящиеся в столбцах типов varchar, nvarchar, varbinary или sql_variant, которые превышают ограничение размера строки, равное 8 060 байт. Страницы имеют тип Text/Image.

Дополнительные сведения о типах страниц см. в разделе [Страницы и экстенды](#).

Куча или сбалансированное дерево могут иметь лишь одну единицу распределения каждого типа в отдельной секции. Для просмотра сведений о единице распределения таблицы или индекса можно воспользоваться представлением каталога [sys.allocation\\_units](#).

## Единица распределения IN\_ROW\_DATA

Для каждой секции, используемой таблицей (кучей или кластеризованной таблицей), индексом или индексированным представлением, существует одна единица распределения IN\_ROW\_DATA, которая состоит из коллекции страниц данных. Эта единица распределения также содержит дополнительные коллекции страниц для реализации каждого некластеризованного индекса и XML-индекса, определенного для таблицы или представления. Коллекции страниц в каждой секции таблицы, индекса или индексированного представления закреплены указателями страницы в системном представлении **sys.system\_internals\_allocation\_units**.

### Важно!

Системное представление **sys.system\_internals\_allocation\_units** зарезервировано только для внутреннего использования Microsoft SQL Server. Совместимость с будущими версиями не гарантируется.

Каждой секции таблицы, индекса и индексированного представления отвечает одна строка в представлении **sys.system\_internals\_allocation\_units**, уникально определенная идентификатором контейнера (**container\_id**). Идентификатор контейнера однозначно сопоставлен с идентификатором **partition\_id** в представлении каталога **sys.partitions**, которое поддерживает связь между данными таблицы, индекса или индексированного представления, хранящимися в секции, и единицами распределения, используемыми для управления данными внутри секции.

Размещение страниц в секции таблицы, индекса или индексированного представления управляется цепочкой IAM-страниц.

Столбец **first\_iam\_page** представления **sys.system\_internals\_allocation\_units** указывает на первую IAM-страницу в цепочке IAM-страниц, управляющую пространством, выделенным для таблицы, индекса или индексированного представления в единице распределения **IN\_ROW\_DATA**.

Представление каталога **sys.partitions** возвращает по одной строке для каждой секции в таблице или индексе.

- Куча имеет строку в представлении каталога **sys.partitions** с **index\_id** = 0.  
Столбец **first\_iam\_page** в представлении **sys.system\_internals\_allocation\_units** указывает на цепочку IAM для коллекции страниц данных кучи в указанной секции. Сервер использует IAM-страницы, чтобы найти страницы в коллекции страниц данных, так как они не связаны.
- Кластеризованный индекс для таблицы или представления имеет одну строку в представлении **sys.partitions** с **index\_id** = 1.  
Столбец **root\_page** в представлении **sys.system\_internals\_allocation\_units** указывает на вершину сбалансированного дерева кластеризованного индекса в указанной секции. Сервер использует индекс сбалансированного дерева для поиска страниц данных в секции.
- Каждый некластеризованный индекс, созданный для таблицы или представления, имеет одну строку в представлении **sys.partitions** с **index\_id** > 1.  
Столбец **root\_page** в представлении **sys.system\_internals\_allocation\_units** указывает на вершину сбалансированного дерева некластеризованного индекса в указанной секции.
- Каждая таблица, в которой есть по крайней мере один столбец LOB, имеет строку в представлении каталога **sys.partitions** с **index\_id** > 250.  
Столбец **first\_iam\_page** указывает на цепочку IAM-страниц, которая управляет страницами в единице распределения **LOB\_DATA**.

## Единица распределения **ROW\_OVERFLOW\_DATA**

Для каждой секции, используемой таблицей (кучей или кластеризованной таблицей), индексом или индексированным представлением, существует единица распределения **ROW\_OVERFLOW\_DATA**. Эта единица распределения содержит ноль (0) страниц до тех пор, пока строка данных со столбцами переменной длины (**varchar**, **nvarchar**, **varbinary** или **sql\_variant**) в единице распределения **IN\_ROW\_DATA** не превышает ограничение размера строки 8 КБ. По достижении границы размера SQL Server перемещает столбец с наибольшей шириной из данной строки на страницу в единице распределения **ROW\_OVERFLOW\_DATA**. Указатель на эти внестрочные данные, имеющий длину 24 бита, сохраняется на начальной странице.

Страницы **Text/Image** в единице распределения **ROW\_OVERFLOW\_DATA** управляются таким же образом, что и страницы в единице распределения **LOB\_DATA**. Таким образом, страницы **Text/Image** управляются цепочкой IAM-страниц.

## Единица распределения **LOB\_DATA**

Если таблица или индекс имеют несколько типов данных LOB, для управления хранением данных выделяется по одной единице распределения **LOB\_DATA** для каждой секции.

### Типы данных LOB

включают text, ntext, image, xml, varchar(max), nvarchar(max), varbinary(max) или определяемые пользователем типы CLR (CLR UDT).

## Пример секции и единицы распределения

Следующий пример возвращает секцию и данные единицы распределения для двух таблиц: DatabaseLog, куча с данными LOB и некластеризованными индексами, и Currency, кластеризованная таблица без данных LOB и одним некластеризованным индексом. Обе таблицы имеют единственную секцию.

```
USE Adventureworks2008R2;
GO
SELECT o.name AS table_name, p.index_id, i.name AS index_name , au.type_desc AS
allocation_type, au.data_pages, partition_number
FROM sys.allocation_units AS au
    JOIN sys.partitions AS p ON au.container_id = p.partition_id
    JOIN sys.objects AS o ON p.object_id = o.object_id
    JOIN sys.indexes AS i ON p.index_id = i.index_id AND i.object_id =
p.object_id
WHERE o.name = N'DatabaseLog' OR o.name = N'Currency'
ORDER BY o.name, p.index_id;
```

Ниже приведен результирующий набор. Обратите внимание на то, что таблица DatabaseLog использует все три типа единиц распределения, так как она содержит страницы типов Data и Text/Image. Таблица Currency не содержит данных LOB, но имеет единицу распределения, необходимую для управления страницами данных. Если таблица Currency будет изменена и будет включать столбец типа данных LOB, то будет создана единица распределения LOB\_DATA для управления этими данными.

table_name	index_id	index_name	allocation_type	data_pages	
partition_number					
Currency	1	PK_Currency_CurrencyCode	IN_ROW_DATA	1	1
Currency	3	AK_Currency_Name	IN_ROW_DATA	1	1
DatabaseLog	0	NULL	IN_ROW_DATA	160	1
DatabaseLog	0	NULL	ROW_OVERFLOW_DATA	0	1
DatabaseLog	0	NULL	LOB_DATA	49	1

(5 row(s) affected)

## Структуры кучи

Кучей является таблица без кластеризованного индекса. Для каждой кучи существует одна строка в представлении [sys.partitions](#) с `index_id = 0` для каждой секции, используемой кучей. По умолчанию у кучи есть одна секция. Если куча имеет несколько секций, каждая из них имеет структуру кучи, содержащую данные для этой определенной секции. Например, если у кучи четыре секции, имеются четыре структуры кучи, по одной на каждую секцию.

В зависимости от типов данных в куче, каждая структура кучи имеет одну или несколько единиц распределения для хранения и управления данными определенной секции. У каждой кучи есть, по крайней мере, одна единица распределения

IN\_ROW\_DATA на каждую секцию. У кучи также будет одна единица распределения LOB\_DATA на каждую секцию, если в этой секции есть столбцы больших объектов (LOB). Кроме того, у кучи будет одна единица распределения ROW\_OVERFLOW\_DATA на каждую секцию, если в этой секции есть столбцы переменной длины, с длиной, превышающей предельное значение размера строки в 8060 байт. Дополнительные сведения о единицах распределения см. в разделе [Организация таблиц и индексов](#).

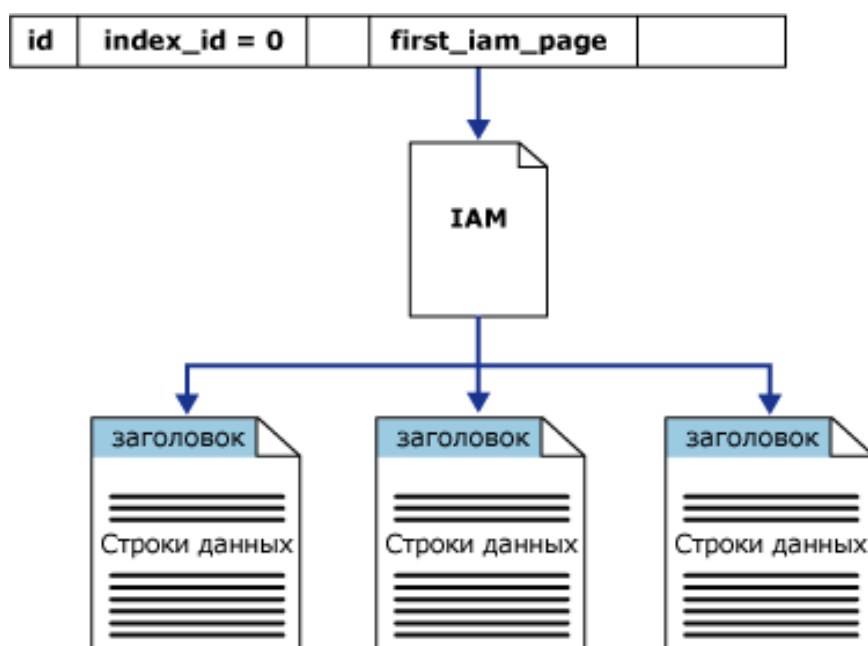
Столбец **first\_iam\_page** в системном представлении **sys.system\_internals\_allocation\_units** указывает на первую IAM-страницу в цепи IAM-страниц, управляющей выделением пространства куче в определенной секции. SQL Server использует IAM-страницы для перемещения по куче. Страницы данных и строки в этих страницах не расположены в каком-либо порядке и не связаны. Единственным логическим соединением страниц данных являются данные, записанные в IAM-страницы.

**Важно!**

Системное представление **sys.system\_internals\_allocation\_units** зарезервировано только для внутреннего использования Microsoft SQL Server. Совместимость с будущими версиями не гарантируется.

Просмотр таблиц или последовательное считывание в куче может выполняться просмотром IAM-страниц для нахождения экстенстов, хранящих страницы кучи. Так как карта IAM представляет экстенсты в том же порядке, в котором они существуют в файлах данных, это означает, что последовательный просмотр кучи выполняется последовательно в каждом файле. Использование IAM-страниц для определения последовательности просмотра означает также, что строки из кучи обычно возвращаются не в том порядке, в котором они вставлялись.

На следующей иллюстрации демонстрируется, как компонент SQL Server Database Engine использует IAM-страницы для получения строк данных из кучи с одной секцией.



## Структуры кластеризованного индекса

В SQL Server индексы организованы в виде сбалансированных деревьев. Каждая страница в сбалансированном дереве индекса называется узлом индекса. Верхний узел сбалансированного дерева называется корневым. Узлы нижнего уровня индекса называются конечными. Все уровни индекса между корневыми и конечными узлами называются промежуточными. В кластеризованном индексе конечные узлы содержат страницы данных базовой таблицы. На страницах индекса корневого и промежуточного узлов находятся строки индекса. Каждая строка индекса содержит ключевое значение и указатель либо на страницу промежуточного уровня сбалансированного дерева, либо на строку данных на конечном уровне индекса. Страницы на каждом уровне связаны в двунаправленный список.

Для каждого кластеризованного индекса таблица [sys.partitions](#) содержит одну строку со значением **index\_id** равным 1 для каждой секции индекса. По умолчанию кластеризованный индекс занимает одну секцию. Если кластеризованный индекс занимает несколько секций, каждая секция содержит сбалансированное дерево, содержащее данные этой секции. Например, если кластеризованный индекс занимает четыре секции, существует четыре сбалансированных дерева: по одному в каждой секции.

В зависимости от типов данных, каждая структура кластеризованного индекса состоит из одной или более единиц распределения, которые применяются для хранения и управления данными секции. Для каждой секции кластеризованный индекс содержит, как минимум, одну единицу распределения IN\_ROW\_DATA. Для хранения столбцов больших объектов (LOB) кластеризованному индексу требуется одна единица распределения LOB\_DATA для каждой секции. Кроме того, для хранения строк переменной длины, превышающих ограничение на размер строки, равное 8 060 байтам, для каждой секции требуется одна единица распределения ROW\_OVERFLOW\_DATA. Дополнительные сведения о единицах распределения см. в разделе [Организация таблиц и индексов](#).

Страницы в цепочке данных и строки, которые они содержат, упорядочены по значению ключа кластеризованного индекса. Все строки вставляются так, чтобы значение ключа составляло вместе с существующими строками упорядоченную последовательность. Коллекции страниц сбалансированного дерева закреплены указателями в системном представлении **sys.system\_internals\_allocation\_units**.

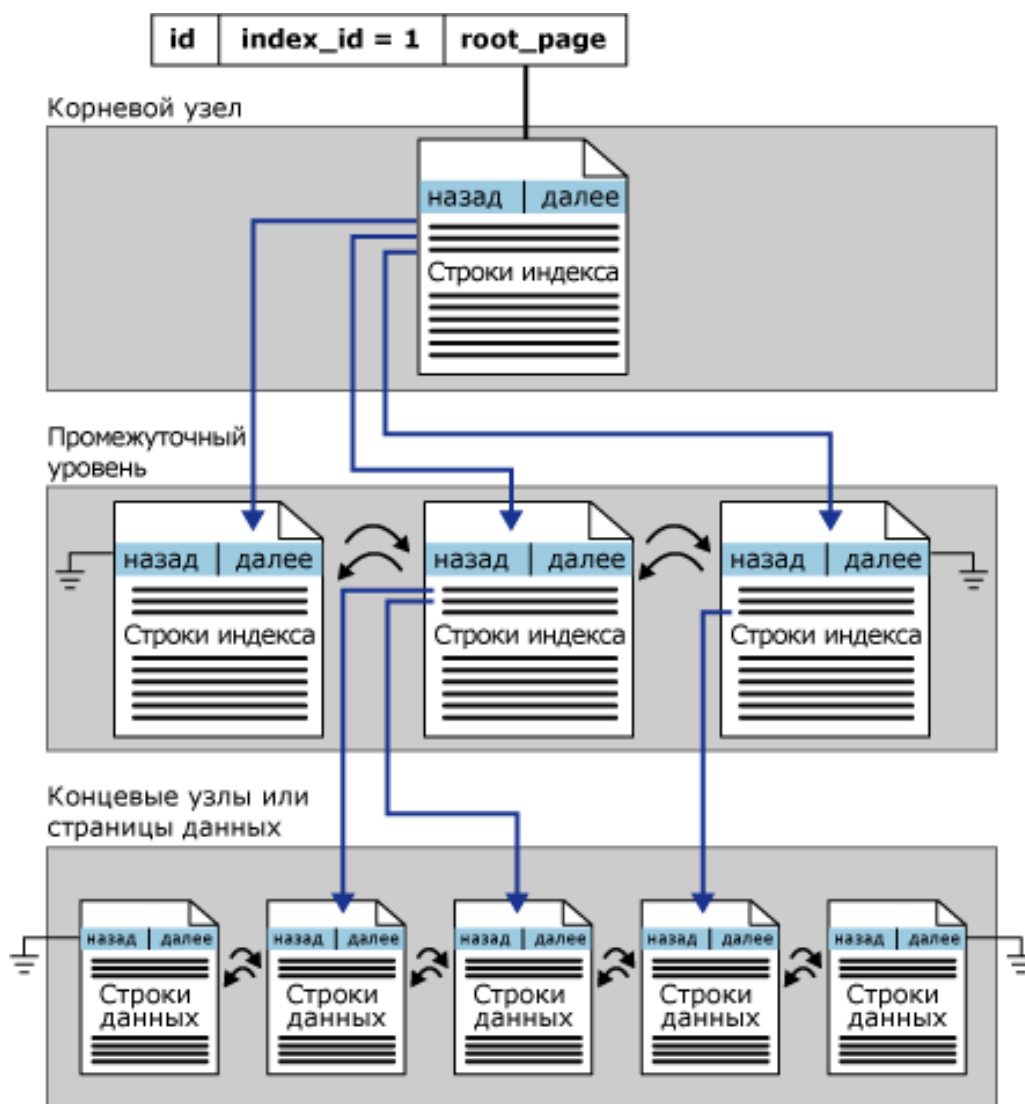
### Важно!

Системное представление **sys.system\_internals\_allocation\_units** зарезервировано только для внутреннего использования Microsoft SQL Server. Совместимость с будущими версиями не гарантируется.

Столбец **root\_page** в таблице **sys.system\_internals\_allocation\_units** содержит указатели на корневые узлы кластеризованного индекса для каждой секции. SQL Server движется вниз по индексу, чтобы найти строку, соответствующую ключу кластеризованного индекса. Чтобы найти диапазон ключей, SQL Server сначала находит начальное значение ключа в диапазоне, а затем сканирует страницы данных, используя

указатели на следующую и предыдущую страницу. Чтобы найти первую страницу в цепочке страниц данных, SQL Server движется по самым левым указателям от корня индекса.

На следующем рисунке изображена структура кластеризованного индекса для одной секции.



## Структуры некластеризованных индексов

Некластеризованные индексы имеют ту же структуру сбалансированного дерева, что и кластеризованные индексы; существуют только следующие различия:

- строки данных в базовой таблице не сортируются и хранятся в порядке, который основан на их некластеризованных ключах;
- конечный уровень некластеризованного индекса состоит из страниц индекса вместо страниц данных.

Некластеризованные индексы могут определяться на таблице или представлении с кластеризованным индексом, или на куче. Каждая строка некластеризованного индекса содержит некластеризованное ключевое значение и указатель на строку. Этот указатель



определяет строку данных кластеризованного индекса или кучи, содержащую ключевое значение.

Указатели строк на строках некластеризованных индексов являются либо указателем на строку, либо ключом кластеризованного индекса для строки, как описано ниже.

- Если таблица является кучей, что означает, что она не содержит кластеризованный индекс, то указатель строки является указателем на строку. Указатель строки строится на основе идентификатора файла (ID), номера страницы и номера строки на странице. Весь указатель целиком называется идентификатором строки (RID).
- Если для таблицы имеется кластеризованный индекс или индекс построен на индексированном представлении, то указатель строки — это ключ кластеризованного индекса для строки. Если кластеризованный индекс не является уникальным индексом, то SQL Server делает все имеющиеся повторяющиеся ключи уникальными путем добавления внутри созданного значения, называемого **uniqueifier**. Это четырех байтовое значение невидимо для пользователей. Оно используется тогда, когда необходимо сделать кластеризованный ключ уникальным, чтобы использовать в некластеризованных индексах. SQL Server получает строку данных путем поиска по кластеризованному индексу, используя ключ кластеризованного индекса, который хранится в конечной строке некластеризованного индекса.

Для некластеризованных индексов есть одна строка в таблице [sys.partitions](#) со значением столбца **index\_id** >0 для каждой секции, используемой индексом. По умолчанию некластеризованный индекс включает одну секцию. Если некластеризованный индекс состоит из нескольких секций, то каждая секция имеет структуру сбалансированного дерева, в которой содержатся индексные строки для данной конкретной секции. Например, если некластеризованный индекс состоит из четырех секций, то существуют четыре структуры сбалансированного дерева, по одной на каждую секцию.

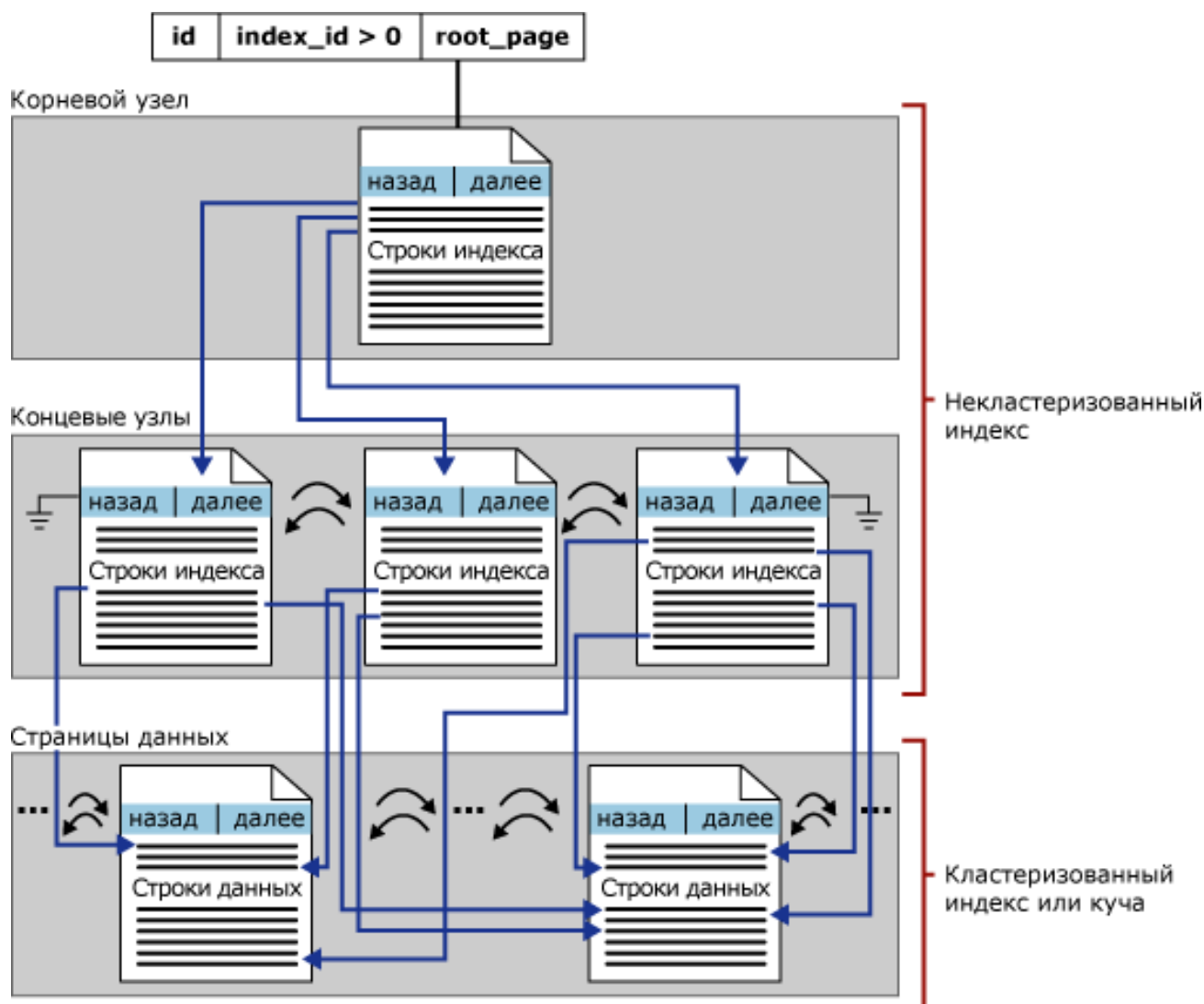
В зависимости от типов данных в некластеризованном индексе каждая структура некластеризованного индекса будет содержать одну или более единиц распределения, в которых хранятся данные для определенной секции. Каждый некластеризованный индекс будет содержать по меньшей мере одну единицу распределения **IN\_ROW\_DATA** на секцию, в которой хранятся страницы сбалансированного дерева индекса.

Некластеризованный индекс будет также содержать одну единицу распределения **LOB\_DATA** на секцию, если в индексе содержатся столбцы типа большого объекта (LOB). Кроме того, некластеризованный индекс будет содержать одну единицу распределения **ROW\_OVERFLOW\_DATA** на секцию, если в индексе содержатся столбцы переменной длины, в которых превышает максимальный размер строки, равный 8 060 байт. Дополнительные сведения о единицах распределения см. в разделе [Организация таблиц и индексов](#). Коллекции страниц сбалансированных деревьев прикрепляются указателями **root\_page** в системном представлении **sys.system\_internals\_allocation\_units**.

#### Важно!

Системное представление **sys.system\_internals\_allocation\_units** зарезервировано только для внутреннего использования Microsoft SQL Server. Совместимость с будущими версиями не гарантируется.

На следующей иллюстрации показана структура некластеризованного индекса, состоящего из одной секции.



### Индексы с включенными столбцами

Функциональность некластеризованных индексов может быть расширена путем добавления включенных столбцов, также называемых неключевыми столбцами, на конечный уровень индекса. Ключевые столбцы хранятся на всех уровнях некластеризованного индекса, тогда как неключевые столбцы хранятся только на конечном уровне. Дополнительные сведения см. в разделе [Индекс с включенными столбцами](#).

## Архитектура обработчика запросов

Компонент Database Engine обрабатывает запросы к различным архитектурам хранения данных, таким как локальные таблицы, секционированные таблицы и таблицы, распределенные по нескольким серверам. В следующих разделах описано, как SQL Server обрабатывает запросы и оптимизирует повторное использование запроса через

кэширование плана выполнения.

## Обработка инструкций SQL

Обработка одиночной инструкции SQL — наиболее распространенный способ, с помощью которого SQL Server выполняет инструкции SQL. Шаги, используемые для обработки одиночной инструкции SELECT, которая обращается только к таблицам локальной базы (а не к представлениям и не к удаленным таблицам), иллюстрируют основной процесс.

## Оптимизация инструкций SELECT

Инструкция SELECT является непроцедурной; она не определяет точные шаги, которые сервер базы данных должен предпринять, для получения запрошенных данных. Это означает, что сервер базы данных должен проанализировать инструкцию для определения самого эффективного способа извлечения запрошенных данных. Это упоминается как оптимизация инструкции SELECT. Выполняющий это компонент называют оптимизатором запросов. Входные данные оптимизатора состоят из самого запроса, схемы базы данных (определений таблиц и индексов) и статистики базы данных. Выходные данные оптимизатора — *план выполнения запроса*, иногда называемый *планом запроса* или просто *планом*. Содержимое плана запроса описывается более подробно далее в этом разделе.

Входные и выходные данные оптимизатора запросов при оптимизации одиночной инструкции SELECT проиллюстрированы в следующей диаграмме:



Инструкция SELECT определяет только следующее.

- Формат результирующего набора. Он указан, главным образом, в списке выбора. Однако другие предложения, например ORDER BY и GROUP BY, также затрагивают конечную форму результирующего набора.
- Таблицы, которые содержат исходные данные. Указываются в предложении FROM.
- Как таблицы логически связаны с целями инструкции SELECT. Это определяется в спецификациях соединения, которые могут появляться в предложении WHERE или в предложении ON, следующем за предложением FROM.
- Условия, которым строки в исходных таблицах должны соответствовать для выбора их инструкцией SELECT. Указываются в предложениях WHERE и HAVING.

План выполнения запроса представляет собой определение следующего.

- Последовательности, в которой происходит обращение к исходным таблицам.

Как правило, существует много последовательностей, в которых сервер базы данных может обращаться к базовым таблицам для построения результирующего набора. Например, если инструкция SELECT ссылается на три таблицы, сервер базы данных сначала может обратиться к **TableA**, использовать данные из **TableA** для извлечения соответствующих строк из **TableB** и затем использовать данные из **TableB** для извлечения данных из **TableC**. Другие последовательности, в которых сервер базы данных может обращаться к таблицам:

**TableC, TableB, TableA** или  
**TableB, TableA, TableC** или  
**TableB, TableC, TableA** или  
**TableC, TableA, TableB**

•Методы, используемые для извлечения данных из каждой таблицы.

Есть различные методы для обращения к данным в каждой таблице. Если необходимы только несколько строк с определенными ключевыми значениями, то сервер базы данных может использовать индекс. Если необходимы все строки в таблице, то сервер базы данных может пропустить индексы и выполнить просмотр таблицы. Если необходимы все строки в таблице, но есть индекс, ключевые столбцы которого находятся в ORDER BY, то выполнение просмотра индекса вместо просмотра таблицы позволит избежать отдельной сортировки результирующего набора. Если таблица является очень маленькой, то просмотры таблицы могут быть самым эффективным методом для практически всех обращений к таблице.

Процесс выбора одного плана выполнения из множества потенциально возможных планов называется оптимизацией. Оптимизатор запросов является одним из самых важных компонентов системы базы данных SQL. Хотя для анализа запроса и выбора плана оптимизатору запросов требуются некоторые накладные расходы, эти накладные расходы обычно многократно окупаются, когда оптимизатор запроса выбирает эффективный план выполнения. Например, двум строительным компаниям могут быть предоставлены идентичные проекты дома. Если одна компания потратит сначала несколько дней на планирование того, как она будет строить дом, а другая компания начнет строить без планирования, то компания, которая потратит время на планирование проекта, вероятно, закончит первой.

Оптимизатор запросов SQL Server основан на оценке стоимости. Каждому возможному плану выполнения соответствует некоторая стоимость, определенная в терминах объема использованных вычислительных ресурсов. Оптимизатор запросов должен проанализировать возможные планы и выбрать один файл с самой низкой предполагаемой стоимостью. Некоторые сложные инструкции SELECT имеют тысячи возможных планов выполнения. В этих случаях оптимизатор запросов не анализирует все возможные комбинации. Вместо этого он использует сложные алгоритмы поиска плана выполнения, имеющего стоимость, близкую к минимальной возможной стоимости.

Оптимизатор запросов SQL Server не выбирает план выполнения, основываясь только на самой низкой стоимости ресурсов; он выбирает такой план, который возвращает результаты пользователю при разумной стоимости ресурсов, но делает это быстрее других. Например, параллельная обработка запроса обычно использует больше ресурсов, чем его последовательная обработка, но завершает выполнение запроса быстрее. Оптимизатор SQL Server будет использовать параллельный план выполнения для возврата результатов, если это не окажет неблагоприятного влияния на загрузку сервера.

Оптимизатор запросов полагается на статистику распределения, когда он оценивает затраты ресурсов для различных методов извлечения сведений из таблицы или индекса. Статистика распределения хранится для столбцов и индексов. Она указывает селективность значений в определенном индексе или столбце. Например, в таблице, представляющей автомобили, много автомобилей имеют одного производителя, но каждый автомобиль имеет уникальный идентификационный номер транспортного средства (VIN). Индекс по VIN является более селективным, чем индекс по производителям. Если статистика индекса не является текущей, оптимизатор запросов, возможно, не сделает лучший выбор для текущего состояния таблицы. Дополнительные сведения о хранении текущей статистики индекса см. в разделе [Использование статистики для повышения производительности запросов](#).

Оптимизатор запросов важен, потому что дает возможность серверу базы данных вносить динамическую коррекцию в изменяющиеся условия в базе данных, не требуя ввода данных от администратора базы данных или программиста. Это дает возможность программистам сосредоточиться на описании конечного результата запроса. Они могут полагаться на то, что оптимизатор запросов будет создавать эффективный план выполнения с учетом состояния базы данных каждый раз при выполнении инструкции.

## Обработка инструкции SELECT

Основные шаги, используемые SQL Server для обработки одиночной инструкции SELECT, включают следующее:

1. Синтаксический анализатор просматривает инструкцию SELECT и разбивает ее на логические единицы, такие как ключевые слова, выражения, операторы и идентификаторы.
2. Строится дерево запроса, иногда называемое деревом последовательности, с описанием логических шагов, необходимых для преобразования исходных данных в формат, требуемый результирующему набору.
3. Оптимизатор запросов анализирует различные способы, с помощью которых можно обратиться к исходным таблицам. Затем он выбирает ряд шагов, которые возвращают результаты быстрее всего и используют меньше ресурсов. Дерево запроса обновляется для записи этого точного ряда шагов. Конечную, оптимизированную версию дерева запроса называют планом выполнения.
4. Реляционный механизм начинает реализовывать план выполнения. В ходе обработки шагов, требующих данных из базовых таблиц, реляционный механизм запрашивает у подсистемы хранилища передачу данных из набора строк, указанных реляционным механизмом.
5. Реляционный механизм преобразует данные, возвращенные подсистемой хранилища, в заданный для результирующего набора формат и возвращает результирующий набор клиенту.

## Обработка других инструкций

Основные шаги, описанные для обработки инструкции SELECT, применимы к другим инструкциям SQL, таким как INSERT, UPDATE и DELETE. Инструкции UPDATE и DELETE имеют адресатом набор строк, которые будут изменены или удалены. Процесс идентификации этих строк является тем же процессом, что и процесс, используемый для идентификации исходных строк, определяющих результирующий набор инструкции

SELECT. Инструкции UPDATE и INSERT могут обе содержать встроенные инструкции SELECT, предоставляющие значения данных, которые будут обновлены или вставлены.

Даже инструкции языка определения данных (DDL), такие как CREATE PROCEDURE или ALTER TABLE, в конечном счете приводятся к ряду реляционных операций на таблицах системного каталога, а иногда (для ALTER TABLE ADD COLUMN) на таблицах данных.

## Рабочие таблицы

Чтобы выполнить логические операции, указанные в инструкции SQL, реляционному механизму может потребоваться создание рабочей таблицы. Рабочие таблицы — это внутренние таблицы, предназначенные для хранения промежуточных результатов. Они создаются для некоторых запросов GROUP BY, ORDER BY и UNION. Например, если предложение ORDER BY ссылается на столбцы, которые не включены в индексы, реляционному механизму может потребоваться создание рабочей таблицы, чтобы отсортировать результирующий набор в необходимом порядке. Рабочие таблицы также иногда применяются для временного хранения результатов выполнения части плана запроса. Рабочие таблицы строятся в базе данных **tempdb** после того, как они больше не нужны, автоматически удаляются.

## Разрешение представлений

Обработчик запросов SQL Server обращается с индексированными и неиндексированными представлениями по-разному:

- Строки индексированного представления хранятся в базе данных в том же формате, что и таблица. Если оптимизатор запросов решает использовать индексированное представление в плане запроса, оно обрабатывается так же, как базовая таблица.
- В случае неиндексированного представления хранится только его определение, но не строки. Оптимизатор запросов интегрирует логику определения представления в план выполнения, создаваемый для инструкции SQL, ссылающейся на неиндексированное представление.

Решение об использовании индексированного представления принимается оптимизатором запросов SQL Server на основе тех же принципов, что и решение об использовании индекса таблицы. Если данные индексированного представления охватывают всю инструкцию SQL или ее часть, а оптимизатор запросов определит, что использовать индекс представления выгодно, он выберет индекс независимо от того, имеется ли в запросе ссылка на представление по его имени. Дополнительные сведения см. в разделе [Разрешение индексов для представлений](#).

Если инструкция SQL ссылается на неиндексированное представление, средство синтаксического анализа и оптимизатор запросов анализируют исходный код инструкции SQL и представления, разрешая их в один план выполнения. Отдельных планов для инструкции SQL и представления нет.

Рассмотрим следующее представление:

```
USE Adventureworks2008R2;  
GO
```

```
CREATE VIEW EmployeeName AS
SELECT h.BusinessEntityID, p.LastName, p.FirstName
FROM HumanResources.Employee AS h
JOIN Person.Person AS p
ON h.BusinessEntityID = p.BusinessEntityID;
GO
```

Обе следующих инструкции SQL, основанных на данном представлении, выполняют одни и те же операции над базовой таблицей, формируя одинаковый результат:

```
/* SELECT referencing the EmployeeName view. */
SELECT LastName AS EmployeeLastName, SalesOrderID, OrderDate
FROM AdventureWorks2008R2.Sales.SalesOrderHeader AS soh
JOIN AdventureWorks2008R2.dbo.EmployeeName AS EmpN
ON (soh.SalesPersonID = EmpN.BusinessEntityID)
WHERE OrderDate > '20020531';

/* SELECT referencing the Person and Employee tables directly. */
SELECT LastName AS EmployeeLastName, SalesOrderID, OrderDate
FROM AdventureWorks2008R2.HumanResources.Employee AS e
JOIN AdventureWorks2008R2.Sales.SalesOrderHeader AS soh
ON soh.SalesPersonID = e.BusinessEntityID
JOIN AdventureWorks2008R2.Person.Person AS p
ON e.BusinessEntityID = p.BusinessEntityID
WHERE OrderDate > '20020531';
```

Используя входящее в среду Средства SQL Server Management Studio средство отображения планов Showplan, нужно убедиться, что для обеих этих инструкций SELECT ядро СУБД создает один и тот же план выполнения.

### **Использование подсказок с представлениями**

Подсказки, связываемые с представлениями в запросах, могут конфликтовать с другими подсказками, которые обнаруживаются при расширении представления для доступа к его базовым таблицам. Когда это происходит, запрос возвращает ошибку. Взгляните, например, на следующее представление, в определение которого входит табличная подсказка:

```
USE AdventureWorks2008R2;
GO
CREATE VIEW Person.AddrState WITH SCHEMABINDING AS
SELECT a.AddressID, a.AddressLine1,
       s.StateProvinceCode, s.CountryRegionCode
FROM Person.Address a WITH (NOLOCK), Person.StateProvince s
WHERE a.StateProvinceID = s.StateProvinceID;
Предположим, что вводится следующий запрос:
```

```
SELECT AddressID, AddressLine1, StateProvinceCode, CountryRegionCode
FROM Person.AddrState WITH (SERIALIZABLE)
WHERE StateProvinceCode = 'WA';
```

Он завершится ошибкой, поскольку подсказка SERIALIZABLE, примененная в запросе к представлению Person.AddrState, при расширении представления распространится как на таблицу Person.Address, так и на таблицу Person.StateProvince. Однако при расширении представления будет также обнаружена подсказка NOLOCK, связанная с

таблицей Person.Address. Из-за конфликта подсказок SERIALIZABLE и NOLOCK итоговый запрос окажется неправильным.

Между собой конфликтуют табличные подсказки PAGLOCK, NOLOCK, ROWLOCK, TABLOCK и TABLOCKX, а также табличные подсказки HOLDLOCK, NOLOCK, READCOMMITTED, REPEATABLEREAD и SERIALIZABLE.

Подсказки могут распространяться через уровни вложенных представлений.

Предположим, что в запросе подсказка HOLDLOCK применяется к представлению v1. При расширении представления v1 выясняется, что представление v2 является частью его определения. Определение представления v2 включает связанную с одной из его базовых таблиц подсказку NOLOCK. Однако эта таблица также наследует указанную в запросе подсказку HOLDLOCK, примененную к представлению v1. Из-за конфликта подсказок NOLOCK и HOLDLOCK запрос завершится ошибкой.

Если в запросе, включающем представление, используется подсказка FORCE ORDER, порядок соединения таблиц в представлении определяется по позиции представления в конструкции упорядочения. Например, приведенный ниже запрос выбирает данные из трех таблиц и представления:

```
SELECT * FROM Table1, Table2, View1, Table3
WHERE Table1.Col1 = Table2.Col1
      AND Table2.Col1 = View1.Col1
      AND View1.Col2 = Table3.Col2;
OPTION (FORCE ORDER)
```

Допустим, что представление View1 определено следующим образом:

```
CREATE VIEW View1 AS
SELECT Colx, Coly FROM TableA, TableB
WHERE TableA.ColZ = TableB.ColZ;
```

В этом случае порядок соединения таблиц в плане запроса будет таким: Table1, Table2, TableA, TableB, Table3.

## Разрешение индексов для представлений

SQL Server использует индексированное представление, как и индекс, в плане запроса только в том случае, если оптимизатор запросов определит, что получит от этого выигрыш.

Индексированные представления можно создавать в любом выпуске SQL Server. В SQL Server Enterprise оптимизатор запросов учитывает индексированные представления автоматически. Чтобы использовать индексированные представления в любых других выпусках, следует применить табличную подсказку NOEXPAND.

Оптимизатор запросов SQL Server пользуется индексированными представлениями при соблюдении следующих условий:

- Следующие параметры сеанса установлены в значение ON:
  - ANSI\_NULLS
  - ANSI\_PADDING



- ANSI\_WARNINGS
- ARITHABORT
- CONCAT\_NULL\_YIELDS\_NULL
- QUOTED\_IDENTIFIER
- Параметр сеанса NUMERIC\_ROUNDABORT установлен в значение OFF.
- Оптимизатор запросов находит соответствие между столбцами индексируемого представления и элементами запроса, например:
  - предикатами условия поиска в предложении WHERE;
  - операциями соединения;
  - статистическими функциями;
  - предложениями GROUP BY;
  - ссылками на таблицы.
- Предполагаемые затраты на использование индекса имеют меньшую стоимость по сравнению с любыми механизмами доступа, имеющимися в распоряжении оптимизатора запросов.
- Каждая таблица, на которую ссылается запрос (либо прямо, либо при расширении представления для доступа к его базовым таблицам), соответствующая табличной ссылке в индексируемом представлении, должна иметь в запросе точно такой же набор подсказок.

#### Примечание

Подсказки READCOMMITTED и READCOMMITTEDLOCK в данном контексте всегда рассматриваются как разные, независимо от уровня изоляции текущей транзакции.

За исключением требований к параметрам SET и табличным подсказкам, это те же самые правила, при помощи которых оптимизатор запросов выясняет, подходит ли индекс таблицы для выполнения запроса. Для использования индексируемого представления в запросе больше ничего указывать не нужно.

Запрос необязательно должен ссылаться в предложении FROM на индексируемое представление, чтобы оптимизатор запросов его использовал. Если запрос ссылается на столбцы в базовой таблице, которые также присутствуют в индексируемом представлении, и оптимизатор запросов определяет, что индексируемое представление будет иметь самую низкую стоимость механизма доступа, он применит индексируемое представление точно так же, как он применяет индекс базовой таблицы, если на него отсутствуют прямые ссылки в запросе. Оптимизатор запросов может применить представление и в том случае, если оно содержит столбцы, на которые не ссылается запрос, если это представление обеспечивает самую низкую стоимость доступа к одному или нескольким столбцам, указанным в запросе.

Индексируемое представление, указанное в предложении FROM, оптимизатор запросов рассматривает как стандартное представление. В начале процесса оптимизации оптимизатор запросов расширяет определение представления в запрос, а затем выполняет в индексируемом представлении поиск соответствий. В окончательном плане выполнения, выбранном оптимизатором, может быть использовано индексируемое представление, или же план может материализовать необходимые данные, производя доступ к базовым таблицам этого представления. Оптимизатором будет выбран вариант с наименьшей стоимостью.

### *Использование подсказок с индексированными представлениями*

Можно избежать использования в запросе индексов представления, указав подсказку в запросе `EXPAND VIEWS`, либо при помощи табличной подсказки `NOEXPAND` принудительно задействовать индекс для индексированного представления, указанного в запросе в предложении `FROM`. Однако оптимизатору запросов следует разрешить динамически определять лучший метод доступа для каждого из запросов. Ограничьте применение подсказок `EXPAND` и `NOEXPAND` только теми случаями, когда очевидно, что они значительно повысят производительность.

Параметр `EXPAND VIEWS` указывает, что оптимизатор запросов не будет использовать индексы представления для всего запроса.

Если для представления указана подсказка `NOEXPAND`, оптимизатор запросов предполагает использование всех индексов, определенных в представлении. Подсказка `NOEXPAND` может иметь необязательное предложение `INDEX()`, которое заставляет оптимизатор запросов пользоваться указанными индексами. Она может быть указана только для индексированного представления и не применяется для представлений, не имеющих индексов.

Если в запросе, содержащем представление, не заданы ни `NOEXPAND`, ни `EXPAND VIEWS`, это представление расширяется для доступа к базовым таблицам. Если запрос представления содержит какие-либо табличные подсказки, они распространяются на базовые таблицы. (Этот процесс подробно описан в разделе [Разрешение представлений](#).) Пока подсказки, имеющиеся в базовых таблицах представления, идентичны, для запроса может устанавливаться соответствие с индексированным представлением. Чаще всего эти подсказки соответствуют друг другу, поскольку они наследуются непосредственно из представления. Однако если запрос ссылается на таблицы, а не на представления, и применяемые к этим таблицам подсказки неидентичны, то для такого запроса соответствие с индексированным представлением устанавливаться не может. Если подсказки `INDEX`, `PAGLOCK`, `ROWLOCK`, `TABLOCKX`, `UPDLOCK` или `XLOCK` применяются к таблицам, на которые запрос ссылается после расширения представления, для этого запроса не может быть установлено соответствие с индексированным представлением.

Если табличная подсказка вида `INDEX (index_val[ ,...n] )` ссылается на представление в запросе, а подсказка `NOEXPAND` не указана, подсказка индекса не обрабатывается. Конкретный индекс следует указывать через `NOEXPAND`.

Обычно, если оптимизатор запросов устанавливает соответствие индексированного представления запросу, все заданные в таблицах или представлениях запроса подсказки применяются непосредственно к индексированному представлению. Если оптимизатор запросов решил не использовать индексированное представление, все подсказки распространяются непосредственно на таблицы, на которые ссылается это представление. Дополнительные сведения см. в разделе [Разрешение представлений](#). Это не относится к подсказкам соединения. Они применяются только в той исходной позиции запроса, где они указаны. Подсказки в соединении оптимизатором запросов при установке соответствия запроса индексированным представлениям не рассматриваются. Если план запроса использует индексированное представление,

которое совпадает с частью запроса, содержащей подсказку соединения, последняя в данном плане не используется.

В SQL Server 2008 в определении индексированных представлений подсказки не допускаются. В режимах совместимости 80 и выше SQL Server пропускает подсказки при работе с определениями индексированных представлений и при выполнении содержащих их запросов. В режиме совместимости 80 использование подсказок в определениях индексированных представлений не вызывает ошибок синтаксиса — они просто пропускаются.

## Разрешение распределенных секционированных представлений

Обработчик запросов SQL Server оптимизирует производительность распределенных секционированных представлений. При оптимизации распределенных секционированных представлений важно минимизировать количество данных, передаваемых между серверами.

SQL Server интеллектуальные динамические планы, которые позволяют эффективно использовать распределенные запросы для доступа к данным в таблицах удаленных серверов.

- Обработчик запросов сначала использует OLE DB для получения определений проверочных ограничений для каждой таблицы сервера. Это позволяет ему определить распределение ключевых значений между таблицами серверов.
- Обработчик запросов сравнивает диапазоны ключей, заданные в предложении WHERE инструкции SQL, со схемой, показывающей, как строки распределены между таблицами серверов. Затем обработчик запросов строит план выполнения, который использует распределенные запросы для получения только тех удаленных строк, которые требуются для завершения инструкции SQL. Кроме того, план выполнения строится таким образом, чтобы обращение к удаленным данным или метаданным выполнялось только в тот момент, когда они требуются.

Например, рассмотрим систему, в которой пользовательская таблица разделена на три секции на серверах **Server1** (**CustomerID** от 1 до 3 299 999), **Server2** (**CustomerID** от 3 300 000 до 6 599 999) и **Server3** (**CustomerID** от 6 600 000 до 9 999 999).

Допустим, план выполнения строится для следующего запроса, который выполняется на сервере **Server1**:

```
SELECT *  
FROM CompanyData.dbo.Customers  
WHERE CustomerID BETWEEN 3200000 AND 3400000;
```

План выполнения этого запроса извлекает строки со значениями ключей **CustomerID** от 3 200 000 до 3 299 999 из локальной таблицы сервера и вызывает распределенный запрос для получения строк со значениями ключей от 3 300 000 до 3 400 000 из **Server2**.

Обработчик запросов сервера SQL Server умеет встраивать динамическую логику в планы выполнения инструкций SQL, когда значения ключей при построении запроса неизвестны. Рассмотрим следующую хранимую процедуру:

```
CREATE PROCEDURE GetCustomer @CustomerIDParameter INT
```

```
AS
SELECT *
FROM CompanyData.dbo.Customers
WHERE CustomerID = @CustomerIDParameter;
```

SQL Server не может предсказать, какое значение ключа будет присвоено параметру **@CustomerIDParameter** при выполнении процедуры. Поскольку значение ключа предсказать нельзя, обработчик запросов не может заранее определить, к какой таблице потребуется доступ. В этом случае SQL Server встраивает в план выполнения условную логику, называемую динамическими фильтрами, для управления доступом к удаленным таблицам на основе значения входного параметра. Если предположить, что хранимая процедура **GetCustomer** была выполнена на сервере **Server1**, логику плана выполнения можно представить следующим образом:

```
IF @CustomerIDParameter BETWEEN 1 and 3299999
    Retrieve row from local table CustomerData.dbo.Customer_33
ELSEIF @CustomerIDParameter BETWEEN 3300000 and 6599999
    Retrieve row from linked table Server2.CustomerData.dbo.Customer_66
ELSEIF @CustomerIDParameter BETWEEN 6600000 and 9999999
    Retrieve row from linked table Server3.CustomerData.dbo.Customer_99
```

Иногда SQL Server строит динамические планы выполнения даже для непараметризованных запросов. Оптимизатор может параметризовать запрос так, чтобы план выполнения можно было использовать повторно. Если оптимизатор параметризует запрос, ссылающийся на секционированное представление, он не будет знать, находятся ли нужные строки в заданной базовой таблице. В дальнейшем ему придется использовать динамические фильтры в планах выполнения. Дополнительные сведения см. в разделе [Простая параметризация](#).

## Выполнение хранимых процедур и триггеров

SQL Server хранит только исходный код хранимых процедур и триггеров. При выполнении хранимой процедуры или триггера в первый раз исходный код компилируется в план выполнения. Если очередной вызов хранимой процедуры или триггера будет инициирован до устаревания плана выполнения, реляционный механизм обнаружит существующий план и использует его повторно. Если план устарел и был удален из памяти, будет создан новый план. Этот процесс похож на то, как SQL Server обрабатывает все инструкции SQL. Основное преимущество хранимых процедур и триггеров SQL Server над пакетами динамического кода SQL в плане быстродействия заключается в том, что их инструкции SQL всегда остаются постоянными. Благодаря этому реляционный механизм может с легкостью сопоставлять их с любыми существующими планами выполнения. Это облегчает повторное использование планов хранимых процедур и триггеров.

Планы выполнения хранимых процедур и триггеров обрабатываются отдельно от плана выполнения пакета, вызвавшего хранимую процедуру или приведшего к срабатыванию триггера. Это способствует повторному использованию планов выполнения хранимых процедур и триггеров.

## Кэширование и повторное использование плана выполнения

В SQL Server есть пул памяти, предназначенный для хранения планов выполнения и буферов данных. Процентное соотношение размера пула, выделенного для планов выполнения и буферов данных, динамически изменяется в зависимости от состояния системы. Часть пула памяти, используемого для хранения планов выполнения, называется кэшем процедур.

В SQL Server планы выполнения состоят из следующих основных компонентов.

- План запроса

Тело плана выполнения является реентерабельной структурой данных только для чтения, которая предназначена для использования любым числом пользователей. Оно называется планом запроса. План запроса не содержит контекста пользователя. В памяти не может находиться более одной или двух копий плана запроса: одна — для всех последовательных выполнений, а другая — для всех параллельных выполнений. Одна параллельная копия обслуживает все параллельные выполнения независимо от степени параллелизма.

- Контекст выполнения

Для каждого пользователя, который в настоящий момент выполняет запрос, имеется структура данных, которая содержит данные, относящиеся к данному выполнению, например значения параметров. Эта структура данных называется контекстом выполнения. Структуры данных контекста выполнения являются повторно используемыми. Если пользователь выполняет запрос и одна из структур не используется, она повторно инициализируется контекстом нового пользователя.



В SQL Server при выполнении инструкции SQL реляционный механизм сначала просматривает кэш процедур, проверяя, нет ли в нем плана выполнения для такой же инструкции. SQL Server повторно использует любой план выполнения, который найдет, снижая издержки на его повторную компиляцию. Если не найдено ни одного существующего плана, SQL Server формирует для этого запроса новый план.

SQL Server реализует эффективный алгоритм поиска существующих планов выполнения для любой инструкции SQL. В большинстве систем ресурсы, затрачиваемые на поиск готового плана, всегда меньше ресурсов, затрачиваемых на повторную компиляцию каждой инструкции SQL.

Алгоритмы поиска соответствия инструкции SQL существующему неиспользуемому

плану выполнения в кэше требуют, чтобы все ссылки на объекты были полными. Например, для первой из следующих инструкций SELECT соответствие существующему плану не будет найдено, а для второго — будет:

```
SELECT * FROM Person;
```

```
SELECT * FROM Person.Person;
```

## Удаление планов выполнения из кэша процедур

Планы выполнения остаются в кэше процедур до тех пор, пока для их хранения остается достаточно памяти. При нехватке памяти компонент Database Engine для определения плана выполнения, который нужно удалить из кэша процедур, использует подход на основе стоимости. Для принятия основанного на стоимости решения компонент Database Engine увеличивает и уменьшает переменную текущей стоимости для каждого плана выполнения, руководствуясь следующими факторами.

Когда пользовательский процесс добавляет в кэш план выполнения, он устанавливает текущую стоимость равной стоимости компиляции исходного запроса. Для нерегламентированных планов выполнения пользовательский процесс устанавливает значение текущей стоимости равным нулю. После этого каждый раз, когда пользовательский процесс ссылается на план выполнения, он сбрасывает текущую стоимость, делая ее равной исходной стоимости компиляции. Для нерегламентированных планов выполнения пользовательский процесс повышает значение текущей стоимости. Для всех планов максимальное значение текущей стоимости равно исходной стоимости компиляции.

Если возникает нехватка памяти, компонент Database Engine реагирует на нее путем удаления планов выполнения из кэша процедур. Чтобы определить, какие планы следует удалить, компонент Database Engine многократно проверяет состояние каждого плана выполнения и удаляет те из них, для которых текущая стоимость равна нулю. План выполнения с нулевой текущей стоимостью не удаляется автоматически при возникновении нехватки памяти; он удаляется только после проверки компонентом Database Engine, если его текущая стоимость равна нулю. При проверке плана выполнения компонент Database Engine приближает текущую стоимость к нулю, уменьшая ее в случае, если запрос в данный момент не использует план.

Компонент Database Engine многократно проверяет планы выполнения, пока не удалит достаточно, чтобы удовлетворить требования к памяти. При нехватке памяти стоимость плана выполнения может увеличиться и уменьшиться несколько раз. Когда нехватка памяти исчезла, компонент Database Engine прекращает уменьшать текущую стоимость неиспользуемых планов выполнения, и все планы выполнения остаются в кэше процедур, даже если их стоимость равна нулю.

Компонент Database Engine использует монитор ресурсов и пользовательские потоки для освобождения памяти, занимаемой кэшем процедур, при возникновении нехватки памяти. Монитор ресурсов и пользовательские потоки могут проверять выполняющиеся параллельно планы, что позволяет уменьшать текущую стоимость для каждого неиспользуемого плана выполнения. Монитор ресурсов удаляет планы выполнения из кэша процедур, если существует глобальная нехватка памяти. Он освобождает память

для принудительного выполнения политик для системной памяти, памяти процессов, памяти пула ресурсов и максимального размера всех кэшей.

Максимальный размер всех кэшей является функцией от размера буферного пула и не может превышать максимальный объем памяти сервера. Дополнительные сведения о настройке максимального объема памяти сервера см. в описании параметра **max server memory** в разделе [sp\\_configure \(Transact-SQL\)](#).

Пользовательские потоки удаляют планы выполнения из кэша процедур, если существует нехватка памяти в одиночном кэше. Они обеспечивают выполнение политик для максимального размера кэша и максимума записей одиночного кэша.

В следующих примерах показывается, какие планы выполнения удаляются из кэша процедур.

- План выполнения часто используется, поэтому его стоимость никогда не принимает значение ноль. Этот план остается в кэше процедур и не удаляется, пока имеется достаточный объем памяти, а его текущая стоимость не равна нулю.
- Нерегламентированный план выполнения вставляется и не используется до возникновения нехватки памяти. Поскольку нерегламентированные планы выполнения инициализируются текущей стоимостью, которая равна нулю, при проверке плана выполнения компонентом Database Engine обнаруживается нулевая текущая стоимость, а этот план удаляется из кэша процедур. Нерегламентированный план выполнения с нулевой текущей стоимостью остается в кэше процедур при наличии достаточного объема памяти.

Чтобы вручную удалить отдельный план выполнения или все планы, используйте команду [DBCC FREEPROCCACHE \(Transact-SQL\)](#).

## Перекомпиляция планов выполнения

Некоторые изменения в базе данных могут привести к тому, что план выполнения при изменении ее состояния станет неэффективным или неправильным. SQL Server обнаруживает изменения, которые могут сделать план выполнения недействительным, и помечает такой план как неправильный. При следующем выполнении данного запроса план должен быть перекомпилирован. План может стать недействительным в следующих случаях.

- Изменены таблица или представления, на которые ссылается запрос (ALTER TABLE или ALTER VIEW).
- Изменены индексы, используемые планом выполнения.
- Обновлена статистика, используемая планом выполнения, сформированная либо явным образом по UPDATE STATISTICS, либо автоматически.
- Удалены индексы, используемые планом выполнения.
- Явно вызвана процедура **sp\_recompile**.
- Частое изменение ключей (инструкциями INSERT или DELETE от пользователей, изменяющих таблицу, на которую ссылается запрос).
- Для таблиц с триггерами: значительный рост числа строк в таблицах **inserted** и **deleted**.
- Выполнение хранимой процедуры с параметром WITH RECOMPILE.

Большинство перекомпиляций необходимы либо для обеспечения правильности работы инструкции, либо для потенциального ускорения работы плана выполнения.

В SQL Server 2000, независимо от того, какая из инструкций пакета вызвала перекомпиляцию, перекомпилируется весь пакет, переданный через хранимую процедуру, триггер, специальный пакет или подготовленную инструкцию. В SQL Server 2005 и более поздних версиях перекомпилируется только та инструкция пакета, которая вызвала перекомпиляцию. Из-за этого отличия счетчики перекомпиляции в SQL Server 2000 и более поздних версиях несовместимы между собой. Кроме этого, в SQL Server 2005 и более поздних версиях больше типов перекомпиляции, что вызвано расширением набора возможностей.

Перекомпиляция на уровне инструкции дает выигрыш в производительности, поскольку в большинстве случаев перекомпиляция небольшого числа инструкций и связанных с этим потерь занимает меньше ресурсов в плане использования процессорного времени и затрат на блокировки. Этих потерь, таким образом, удастся избежать для остальных инструкций пакета, которые в перекомпиляции не нуждаются.

Событие трассировки **SP:Recompile** приложения Приложение SQL Server Profiler выводит сведения о перекомпиляциях на уровне инструкций. В SQL Server 2000 это событие трассировки выдает отчет только о перекомпиляции пакетов. Затем заполняется столбец **TextData** данного события. Таким образом, практика отслеживания **SP:StmtStarting** или **SP:StmtCompleted** для получения текста Transact-SQL, который вызвал перекомпиляцию, существовавшая в SQL Server 2000, больше не актуальна.

Событие трассировки **SQL:StmtRecompile** выводит сведения о перекомпиляциях на уровне инструкций. Оно может применяться для трассировки и отладки перекомпиляций. Событие **SP:Recompile** создается только для хранимых процедур и триггеров, а **SQL:StmtRecompile** — для хранимых процедур, триггеров, специальных пакетов, пакетов, которые выполняются через **sp\_executesql**, подготовленных запросов и динамического SQL.

Столбец **EventSubClass** для событий **SP:Recompile** и **SQL:StmtRecompile** содержит код в виде целого числа, обозначающий причину перекомпиляции. В следующей таблице перечислено значение для каждого из этих кодов.

Значение EventSubClass	Описание
1	Изменение схемы.
2	Изменение статистики.
3	Отложенная компиляция.
4	Изменение параметра SET.
5	Изменение временной таблицы.



6	Изменение удаленного набора строк.
7	Изменение разрешения FOR BROWSE.
8	Изменение среды уведомлений о запросах.
9	Изменение секционированного представления.
10	Изменение параметров курсора.
11	Запрошено OPTION (RECOMPILE).

#### Примечание

Если параметр базы данных `AUTO_UPDATE_STATISTICS` устанавливается в значение `ON`, запросы перекомпилируются в том случае, если они указывают на целевые таблицы или индексированные представления, для которых изменена статистика или мощность которых со времени последнего выполнения изменилась в значительной степени. Это относится к стандартным пользовательским, временным таблицам, а также таблицам **inserted** и **deleted**, созданным триггерами DML. На производительность запроса оказывают влияние и неумеренные перекомпиляции, вызванные установкой этого параметра в значение `OFF`. Если параметр базы данных `AUTO_UPDATE_STATISTICS` установлен в значение `OFF`, перекомпиляции по причине изменения статистики или количества элементов не выполняются за исключением таблиц **inserted** и **deleted** которые созданы триггерами DML `INSTEAD OF`. Так как данные таблицы создаются в базе данных **tempdb**, перекомпиляция запросов, которые обращаются к этим таблицам, зависит от значения параметра `AUTO_UPDATE_STATISTICS` в базе данных **tempdb**. Обратите внимание, что в SQL Server 2000 запросы при изменении количества элементов или статистики продолжают выполняться в таблицах триггеров DML **inserted** и **deleted**, даже если указанный параметр установлен в `OFF`. Дополнительные сведения об отключении параметра `AUTO_UPDATE_STATISTICS` см. в разделе [Использование статистики для повышения производительности запросов](#).

## Параметры и повторное использование планов выполнения

Использование параметров, включая маркеры параметров в приложениях ADO, OLE DB и ODBC, может повысить уровень использования планов выполнения.

#### Примечание по безопасности

Использование параметров и маркеров параметров для хранения введенных конечными пользователями значений является более безопасным, чем сцепление значений в строку, которая затем выполняется методом API доступа к данным, инструкцией `EXECUTE` либо хранимой процедурой `sp_executesql`.

Единственная разница между следующими двумя инструкциями SELECT — в значениях, сравниваемых в предложении WHERE:

```
SELECT *  
FROM AdventureWorks2008R2.Production.Product  
WHERE ProductSubcategoryID = 1;
```

```
SELECT *  
FROM AdventureWorks2008R2.Production.Product  
WHERE ProductSubcategoryID = 4;
```

Единственная разница между планами выполнения для данных запросов — в значении, хранимом для сравнения со столбцом **ProductSubcategoryID**. В то время как выявление факта формирования инструкциями одного и того же плана и повторного его использования является основной задачей SQL Server, SQL Server не всегда может это обнаружить в сложных инструкциях SQL.

Отделение констант от инструкции SQL с помощью параметров помогает реляционному механизму распознавать дубликаты планов. Параметры можно использовать следующими способами.

- В языке Transact-SQL используйте процедуру **sp\_executesql**:

```
DECLARE @MyIntParm INT  
SET @MyIntParm = 1  
EXEC sp_executesql  
    N'SELECT *  
    FROM AdventureWorks2008R2.Production.Product  
    WHERE ProductSubcategoryID = @Parm',  
    N'@Parm INT',  
    @MyIntParm
```

Этот метод рекомендуется для скриптов языка Transact-SQL, хранимых процедур и триггеров, динамически формирующих инструкции SQL.

- В технологиях ADO, OLE DB и ODBC используются маркеры параметров. Маркеры параметров представляют собой знаки вопроса (?), заменяющие константу в инструкции SQL и привязываемые к программной переменной. Например, в приложении ODBC можно сделать следующее:

- использовать параметр **SQLBindParameter** для привязки целочисленной переменной к первому маркеру параметра в инструкции SQL;
- поместить целочисленное значение в переменную;
- выполнить инструкцию, указав маркер параметра (?):

```
SQLExecDirect(hstmt,  
    "SELECT *  
    FROM AdventureWorks2008R2.Production.Product  
    WHERE ProductSubcategoryID = ?",  
    SQL_NTS);
```

Если в приложениях используются маркеры параметров, поставщик OLE DB для собственного клиента SQL Server и драйвер ODBC для собственного клиента SQL Server, включенные в состав SQL Server, используют для отправки инструкций в SQL Server процедуру **sp\_executesql**.

- Чтобы проектировать хранимые процедуры, использующие указанные разработчиком параметры.

Если структура приложения не предусматривает явной подготовки параметров, можно воспользоваться оптимизатором запросов SQL Server для автоматической параметризации некоторых запросов с использованием установленного по умолчанию поведения [Простая параметризация](#). В качестве альтернативы можно заставить оптимизатор запросов учитывать параметризацию всех запросов к базе данных путем установки параметра `PARAMETERIZATION` инструкции `ALTER DATABASE` в значение `FORCED`. Дополнительные сведения см. в разделе [Принудительная параметризация](#).

При включенной принудительной параметризации может также иметь место и простая параметризация. Например, в соответствии с правилами принудительной параметризации следующий запрос не может быть параметризован.

```
SELECT * FROM Person.Address  
WHERE AddressID = 1 + 2;
```

Однако он может быть параметризован согласно правилам простой параметризации. В случае неуспешной попытки принудительной параметризации впоследствии производятся попытки использования простой параметризации.

## Простая параметризация

В SQL Server использование параметров или маркеров параметров в инструкциях Transact-SQL увеличивает возможности реляционного механизма по применению существующих скомпилированных планов выполнения для новых инструкций SQL.

### Примечание по безопасности

Использование параметров или маркеров параметров для хранения значений, введенных конечными пользователями, более безопасно, чем сцепление значений в строку, которая впоследствии выполняется с помощью API-метода доступа к данным, инструкции `EXECUTE` или хранимой процедуры `sp_executesql`.

Если инструкция SQL выполняется без параметров, SQL Server неявно параметризует инструкцию, чтобы увеличить возможность ее противопоставления существующему плану выполнения. Данный процесс называется простой параметризацией. В SQL Server 2000 этот процесс назывался автоматической параметризацией.

Рассмотрим следующую инструкцию.

```
SELECT * FROM AdventureWorks2008R2.Production.Product  
WHERE ProductSubcategoryID = 1;
```

Значение 1 в конце инструкции может быть указано в виде параметра. Реляционный механизм строит план выполнения для данного пакета, как если бы параметр был указан на месте значения 1. При помощи этой простой параметризации SQL Server распознает, что следующие две инструкции формируют, по сути, одинаковый план выполнения, и повторно использует первый план для второй инструкции:

```
SELECT * FROM AdventureWorks2008R2.Production.Product  
WHERE ProductSubcategoryID = 1;
```

```
SELECT * FROM AdventureWorks2008R2.Production.Product  
WHERE ProductSubcategoryID = 4;
```

В процессе обработки сложных инструкций SQL реляционный механизм может с трудом определять, какие выражения могут быть параметризованы. Чтобы увеличить способность реляционного механизма находить сложным инструкциям SQL соответствующие существующие неиспользованные планы выполнения, необходимо явно указать параметры при помощи процедуры `sp_executesql` или маркеров параметров. Дополнительные сведения см. в разделе [Параметры и повторное использование планов выполнения](#).

#### Примечание

При использовании арифметических операторов (+, -, \*, / или %) для явного или неявного преобразования постоянных значений типов данных `int`, `smallint`, `tinyint` или `bigint` в типы данных `float`, `real`, `decimal` или `numeric` SQL Server применяет специальные правила для вычисления типа и точности результатов выражения. Однако эти правила различаются в зависимости от того, параметризован запрос или нет. Таким образом, одинаковые выражения в запросах могут в некоторых случаях давать отличающиеся результаты. Дополнительные сведения см. в разделе [int, bigint, smallint и tinyint \(Transact-SQL\)](#).

При проведении простой параметризации SQL Server по умолчанию параметризует сравнительно небольшой класс запросов. Однако можно указать, чтобы все запросы в базе данных были параметризованы в соответствии с определенными ограничениями, настроив параметр `PARAMETERIZATION` команды `ALTER DATABASE` на `FORCED`. Уменьшая частоту компиляции запросов, эти действия улучшат производительность баз данных, которые испытывают большие объемы параллельных запросов. Дополнительные сведения см. в разделе [Принудительная параметризация](#).

Иначе можно указать параметризацию одного запроса и других, синтаксически равных, но отличающихся значениями параметра, запросов. Дополнительные сведения см. в разделе [Указание механизма параметризации запросов с помощью структур плана](#).

## Принудительная параметризация

Можно переопределить [простую параметризацию](#), используемую по умолчанию в SQL Server, указав, что все инструкции `SELECT`, `INSERT`, `UPDATE` и `DELETE` в базе данных должны быть параметризованы (с учетом некоторых ограничений). Принудительная параметризация активируется путем установки для параметра `PARAMETERIZATION` значения `FORCED` в инструкции [ALTER DATABASE](#). Принудительная параметризация может улучшить производительность некоторых баз данных, сократив частоту выполнения компиляции и перекомпиляции запросов. Базы данных, которым может пойти на пользу принудительная параметризация, — это, как правило, те, которым приходится выполнять большое количество параллельных запросов из источников наподобие приложений торговых точек.

Если параметру `PARAMETERIZATION` присвоено значение `FORCED`, любое литеральное значение, представленное в инструкции `SELECT`, `INSERT`, `UPDATE` или `DELETE`, заявленной в любой форме, преобразуется в аргумент в процессе компиляции запроса. Исключениями являются литералы, представленные в следующих конструкциях запроса.

- Инструкции INSERT...EXECUTE.
- Инструкции в теле хранимых процедур, триггеров или определяемых пользователем функций. SQL Server уже использует повторно планы запросов для этих подпрограмм.
- Подготовленные инструкции, которые уже были параметризованы приложением на стороне клиента.
- Инструкции, содержащие вызовы метода XQuery, где метод представлен в контексте, где его аргументы обычно параметризуются, например в предложении WHERE. Если метод представлен в контексте, где его аргументы не параметризуются, остальная часть инструкции будет параметризована.
- Инструкции внутри курсора Transact-SQL. (Инструкции SELECT внутри курсоров API-интерфейса параметризуются.)
- Устаревшие конструкции запроса.
- Любая инструкция, выполняемая в контексте ANSI\_PADDING или ANSI\_NULLS со значением OFF.
- Инструкции, содержащие более 2 097 литералов, пригодных для параметризации.
- Инструкции, ссылающиеся на переменные, такие как WHERE T.col2 >= @bb.
- Инструкции, содержащие подсказку в запросе RECOMPILE.
- Инструкции, содержащие предложение COMPUTE.
- Инструкции, содержащие предложение WHERE CURRENT OF.

Кроме того, в запросах не параметризуются следующие предложения (следует иметь в виду, что не параметризуются только предложения; другие предложения внутри того же запроса могут оказаться пригодными для принудительной параметризации).

- <select\_list> в любой инструкции SELECT. Сюда входят списки SELECT во вложенных запросах и списки SELECT внутри инструкций INSERT.
  - Инструкции SELECT во вложенных запросах, представленные внутри инструкции IF.
  - Предложения запроса TOP, TABLESAMPLE, HAVING, GROUP BY, ORDER BY, OUTPUT...INTO или FOR XML.
  - Аргументы, прямые или в качестве подвыражений, для OPENROWSET, OPENQUERY, OPENDATASOURCE, OPENXML или для любого оператора FULLTEXT.
  - Аргументы pattern и escape\_character предложения LIKE.
  - Аргумент style предложения CONVERT.
  - Целочисленные константы внутри предложения IDENTITY.
  - Константы, указанные использованием синтаксиса расширения ODBC.
  - Свертываемые выражения, являющиеся аргументами операторов +, -, \*, / и %.
- При определении пригодности для принудительной параметризации SQL Server рассматривает выражение как свертываемое, если верно хотя бы одно из следующих условий.
- В выражении не представлены столбцы, переменные или вложенные запросы.
  - Выражение содержит предложение CASE.

Дополнительные сведения о свертываемых выражениях см. в разделе [Диагностика низкой производительности запросов. Свертка констант и механизм вычисления выражений во время оценки мощности](#).

- Аргументы для предложений подсказок в запросах. Сюда входит аргумент number\_of\_rows подсказки FAST, аргумент number\_of\_processors подсказки MAXDOP и аргумент number подсказки

**MAXRECURSION.**

Параметризация происходит на уровне отдельных инструкций Transact-SQL. Иными словами, параметризуются отдельные инструкции в пакете. После компиляции параметризованный запрос выполняется в контексте пакета, в котором он был изначально заявлен. Если план выполнения для запроса кэширован, можно определить, был ли параметризован запрос, обратившись к столбцу **sql** в динамическом административном представлении [sys.syscacheobjects](#). Если запрос параметризован, имена и типы данных аргументов располагаются перед текстом заявленного пакета в этом столбце, например (**@1 tinyint**). Дополнительные сведения о кэшировании плана запроса см. в разделе [Кэширование и повторное использование плана выполнения](#).

**Примечание**

Имена аргументов произвольны. Пользователи или приложения не должны опираться на какой-либо конкретный порядок именования. Кроме того, в зависимости от версии SQL Server и пакетов обновления могут меняться имена параметров, выбор литералов, подлежащих параметризации, и разбивка параметризованного текста.

**Типы данных аргументов**

Когда SQL Server параметризует литералы, аргументы преобразовываются в следующие типы данных.

- Целочисленные литералы, размер которых в ином случае соответствовал бы типу данных **int**, параметризуются **int**. Большие целочисленные литералы, являющиеся частью предикатов, которые включают в себя любой оператор сравнения (в том числе **<**, **<=**, **=**, **!=**, **>**, **>=**, **!<**, **!>**, **<>**, **ALL**, **ANY**, **SOME**, **BETWEEN** и **IN**), параметризуются **numeric(38,0)**. Большие литералы, не являющиеся частью предикатов, которые включают в себя операторы сравнения, параметризуются в **numeric** с точностью достаточно большой, чтобы поддержать их размер, и с масштабом 0.
- Числовые литералы с фиксированной запятой, являющиеся частью предикатов, которые включают в себя операторы сравнения, параметризуются в **numeric** с точностью 38 и масштабом достаточно большим, чтобы поддержать их размер. Числовые литералы с фиксированной запятой, не являющиеся частью предикатов, которые включают в себя операторы сравнения, параметризуются в **numeric** с точностью и масштабом достаточно большими, чтобы поддержать их размер.
- Числовые литералы с плавающей запятой параметризуются в **float(53)**.
- Строковые литералы не в формате Юникод параметризуются в **varchar(8000)**, если размер литерала не превышает 8 000 символов, и в **varchar(max)**, если он больше 8 000 символов.
- Строковые литералы в формате Юникод параметризуются в **nvarchar(4000)**, если размер литерала не превышает 4 000 символов, и в **nvarchar(max)**, если литерал больше 4 000 символов.
- Двоичные литералы параметризуются в **varbinary(8000)**, если размер литерала не превышает 8 000 байт. Если он больше 8 000 байт, он преобразуется в **varbinary(max)**.
- Денежные литералы параметризуются в **money**.

**Рекомендации по использованию принудительной параметризации**

Устанавливая для параметра **PARAMETERIZATION** значение **FORCED**, примите во



внимание следующие сведения.

- Принудительная параметризация, в сущности, преобразует литеральные константы в запросе в параметры при компиляции запроса. Следовательно, оптимизатор запросов может выбирать не самые оптимальные планы для запросов. В частности, уменьшается вероятность того, что оптимизатор запросов сопоставит запрос с индексированным представлением или индексом по вычисляемому столбцу. Он может также выбирать не самые оптимальные планы для запросов, ориентированных на секционированные таблицы или распределенные секционированные представления. Принудительная параметризация не должна использоваться в средах, в значительной степени опирающихся на индексированные представления и индексы по вычисляемым столбцам. Параметр `PARAMETERIZATION FORCED` должен использоваться только опытными администраторами баз данных и лишь после того, как будет определено, что такое использование не повредит производительности.
- Распределенные запросы, ссылающиеся на более чем одну базу данных, пригодны для принудительной параметризации, если параметр `PARAMETERIZATION` установлен на `FORCED` в базе данных, в контексте которой выполняется запрос.
- Установка параметра `PARAMETERIZATION` на `FORCED` производит очистку всех планов запросов из кэша планов в базе данных за исключением тех, которые компилируются, перекомпилируются или выполняются в настоящий момент. Планы для запросов, которые компилируются или выполняются в момент изменения настроек, параметризуются при следующем выполнении запроса.
- Настройка параметра `PARAMETERIZATION` выполняется в оперативном режиме и не требует монопольных блокировок на уровне базы данных.
- Принудительная параметризация отключается (устанавливается в `SIMPLE`), если уровень совместимости базы данных SQL Server установлен в 80 или если база данных на экземпляре более ранней версии присоединена к экземпляру SQL Server 2005 или более поздней версии.
- Текущая настройка параметра `PARAMETERIZATION` сохраняется при повторном присоединении или восстановлении базы данных.

Можно перекрывать поведение принудительной параметризации, предписав выполнение попытки простой параметризации для отдельного запроса, а также всех остальных запросов с таким же синтаксисом, отличающихся только значениями аргументов.

Справедливо и обратное: можно потребовать выполнения попытки принудительной параметризации для отдельного набора синтаксически эквивалентных запросов, даже если принудительная параметризация в базе данных отключена. В этих целях используются структуры планов. Дополнительные сведения см. в разделе [Указание механизма параметризации запросов с помощью структур плана](#).

#### Примечание

Если параметр `PARAMETERIZATION` имеет значение `FORCED`, то отчеты об ошибках могут отличаться от отчетов, формируемых при простой параметризации: число сообщений об ошибках в некоторых случаях больше, чем при простой параметризации, а номера строк ошибок могут быть выданы неверно.

## Подготовка инструкций SQL

В реляционном механизме SQL Server введена полная поддержка подготовки инструкций SQL перед их выполнением. Если приложению требуется выполнить

инструкцию SQL несколько раз, то оно может использовать API базы данных следующим образом.

- Однократная подготовка инструкции. Инструкция SQL компилируется в план выполнения.
- Ранее скомпилированный план выполнения выполняется каждый раз при необходимости использовать эту инструкцию. Это избавляет от необходимости повторно компилировать инструкцию SQL при каждом последующем выполнении. Подготовка и выполнение инструкций контролируется функциями и методами API. Они не имеют отношения к языку Transact-SQL. Модель подготовки и выполнения инструкций SQL поддерживается поставщиком OLE DB для собственного клиента SQL Server, а также драйвером ODBC для собственного клиента SQL Server. При запросе на подготовку поставщик или драйвер отправляет инструкцию SQL Server с запросом на подготовку инструкции. SQL Server компилирует план выполнения и возвращает его дескриптор поставщику или драйверу. При запросе на выполнение поставщик или драйвер отправляет на сервер запрос на выполнение плана, связанного с этим дескриптором.

В SQL Server подготовленные инструкции нельзя применять для создания временных объектов. Подготовленные инструкции не могут содержать ссылки на системные хранимые процедуры, создающие временные объекты, такие как временные таблицы. Эти процедуры следует выполнять напрямую.

Злоупотребление моделью подготовки и выполнения может отрицательно сказаться на производительности. Если инструкция выполняется только один раз, то для прямого выполнения потребуется только один цикл приема-передачи с сервером. Для подготовки и выполнения инструкции SQL, которая выполняется только один раз, потребуется два таких цикла: один для подготовки и один для выполнения.

Подготовка инструкции более эффективна, если используются маркеры параметров. Например, предположим, что приложение запросило сведения о продукте из образца базы данных **AdventureWorks2008R2**. Эти сведения приложение может извлечь двумя способами.

Приложение может выполнять отдельный запрос по каждому необходимому продукту:

```
SELECT * FROM AdventureWorks2008R2.Production.Product  
WHERE ProductID = 63;
```

Второй способ заключается в следующем.

1. Приложение подготавливает инструкцию, содержащую маркер параметра (?):  
`SELECT * FROM AdventureWorks2008R2.Production.Product  
WHERE ProductID = ?;`
2. Затем оно связывает переменную программы с этим маркером.
3. Каждый раз, когда требуются сведения о продукте, приложение присваивает связанной переменной ключевое значение и выполняет инструкцию.

Второй способ более эффективен, если инструкция выполняется более трех раз.

В SQL Server модель подготовки и выполнения не дает существенного прироста производительности по сравнению с непосредственным выполнением из-за того, каким образом SQL Server повторно использует планы выполнения. В SQL Server предусмотрены эффективные алгоритмы для сопоставления текущих инструкций SQL и планов, созданных для предыдущих случаев выполнения той же инструкции SQL. Если приложение выполняет инструкцию SQL с маркерами параметров несколько раз, то SQL



Server будет в дальнейшем использовать уже готовый план выполнения (если только этот план не будет удален из кэша процедур). Впрочем, у модели подготовки и выполнения есть следующие достоинства:

- поиск плана производится путем идентификации дескриптора, что эффективнее алгоритмов, которые применяются для сопоставления инструкции SQL и существующих планов выполнения;
- приложение может управлять временем создания и повторного использования плана выполнения;
- Модель подготовки и выполнения можно переносить в другие базы данных, включая более ранние версии SQL Server.

## Параллельная обработка запросов

SQL Server обеспечивает параллельную обработку запросов, оптимизирующую выполнение запросов и операции с индексами на компьютерах, где установлено несколько микропроцессоров (ЦП). Благодаря возможностям параллельной обработки запроса и операций с индексами при помощи нескольких потоков операционной системы SQL Server выполняет эти операции быстрее и эффективнее.

Во время оптимизации запроса SQL Server пытается обнаружить запросы и операции с индексами, которые можно ускорить за счет параллельного выполнения. Для таких запросов SQL Server вставляет в план выполнения операторы обмена, чтобы подготовить запрос к параллельной обработке. Операторы обмена служат для управления процессом, перераспределения данных и управления потоком. К ним относятся логические операторы [Distribute Streams](#), [Repartition Streams](#) и [Gather Streams](#) (в качестве подтипов), один или несколько из которых появляются в выводе инструкции Showplan плана запроса для параллельного запроса.

После вставки операторов обмена получается план параллельного выполнения запроса. План параллельного выполнения запроса может использовать несколько потоков. План последовательного выполнения, который используется для обработки непараллельных запросов, использует только один поток. Фактическое количество потоков параллельного выполнения запроса определяется при инициализации плана выполнения запроса и зависит от сложности и степени параллелизма плана. Степень параллелизма определяет максимальное количество используемых ЦП, а не количество используемых потоков. Степень параллелизма устанавливается на уровне сервера и изменяется системной хранимой процедурой **sp\_configure**. Это значение можно переопределить для отдельных инструкций запроса или индекса при помощи подсказки в запросе MAXDOP или параметра индекса MAXDOP.

Оптимизатор запросов SQL Server не использует план параллельного выполнения для запроса, если выполняется любое из следующих условий:

- Затраты на последовательное выполнение запроса не настолько высоки, чтобы альтернативой ему считался план параллельного выполнения.
- План последовательного выполнения признан более быстрым, чем любой другой возможный план параллельного выполнения данного запроса.
- Запрос содержит скалярные или реляционные операторы, параллельное выполнение которых невозможно. Определенные операторы могут привести к

выполнению участка запроса или всего плана целиком в последовательном режиме.

### **Степень параллелизма**

SQL Server автоматически обнаруживает высшую степень параллелизма для каждого экземпляра параллельного выполнения запроса или индекс операции языка DDL. Это осуществляется на основе следующих критериев.

1. Работает ли SQL Server на компьютере, имеющем более одного микропроцессора или ЦП (таком как симметричный многопроцессорный компьютер (SMP)).

Использовать параллельные запросы могут только компьютеры, имеющие более одного ЦП.

2. Достаточно ли доступных потоков.

Каждый запрос или операция с индексами требуют определенного числа потоков, подлежащих выполнению. Для выполнения параллельного плана требуется больше потоков, чем для выполнения последовательного плана, и число запрашиваемых потоков возрастает по мере увеличения степени параллелизма. Когда требование к потокам параллельного плана для определенной степени параллелизма не может быть удовлетворено, компонент Database Engine уменьшает степень параллелизма автоматически или полностью отказывается от параллельного плана в указанном контексте рабочей нагрузки. В таком случае начинается выполнение последовательного плана (один поток).

3. Тип выполняемого запроса или операции с индексами.

Операции с индексами, которые создают или перестраивают индекс или удаляют кластеризованный индекс и запросы, интенсивно использующие циклы ЦП, являются лучшими кандидатами для параллельного плана. Например, хорошими кандидатами являются соединения больших таблиц, больших статистических выражений и сортировка больших результирующих наборов. Простые запросы, часто находящиеся в приложениях обработки транзакций, находят дополнительную координацию, запрашиваемую для выполнения запроса в параллельном перевешивании возможного повышения производительности. Чтобы отличить запросы, которые выигрывают от параллелизма, и запросы, которые не выигрывают, компонент Database Engine сравнивает предполагаемую стоимость выполняемого запроса или операции с индексами со значением [cost threshold for parallelism](#). Несмотря на то, что это не рекомендуется, пользователи могут менять значение по умолчанию 5 при помощи процедуры [sp\\_configure](#).

4. Достаточно ли число строк, подлежащих обработке.

Если оптимизатор запросов устанавливает, что число строк слишком мало, то для распространения строк он не вставляет операторы преобразования валюты. Следовательно, операторы обрабатываются последовательно. Обработка операторов в последовательном плане позволяет избежать сценариев, когда стоимость запуска, распределения и координации превышает преимущества, достигнутые параллельной обработкой оператора.

5. Доступна ли статистика распределения.

Если наивысшая степень параллелизма невозможна, более низкие степени рассматриваются до того, как отвергается параллельный план.

Например, статистика распределения не может вычисляться при создании кластеризованного индекса на представлении, потому что кластеризованный индекс еще не существует. В таком случае компонент Database Engine не может предоставить наивысшую степень параллелизма для операции с индексами. Однако некоторые операторы, такие как сортировка и сканирование, по-прежнему могут выигрывать от параллельной обработки.

Параллельные операции с индексами доступны только в выпусках SQL Server Developer Edition, Evaluation Edition и Enterprise Edition.

Во время выполнения компонент Database Engine устанавливает, разрешены ли описанные ранее текущая рабочая нагрузка системы и конфигурация для параллельного выполнения. Если параллельное выполнение гарантировано, компонент Database Engine устанавливает оптимальное число потоков и распределяет выполнение параллельного плана по этим потокам. Когда запрос или операция с индексами начинает выполнение на нескольких потоках для параллельного выполнения, такое же число потоков используется до тех пор, пока операция не будет завершена. Компонент Database Engine перепроверяет оптимальное число решений потоков всякий раз при получении плана выполнения из кэша процедуры. Например, одно выполнение запроса может привести к использованию последовательного плана, последующее выполнение того же запроса может привести к параллельному плану, использующему три потока, а третье выполнение может привести к параллельному плану, использующему четыре потока.

В плане параллельного выполнения запроса операторы вставки, обновления и удаления обрабатываются последовательно. Однако предложение WHERE инструкции UPDATE или DELETE или часть SELECT инструкции INSERT могут обрабатываться параллельно. В таком случае изменения фактических данных последовательно применяются к базе данных.

Статические курсоры и курсоры, управляемые набором ключей, могут быть заполнены параллельными планами выполнения. Однако поведение динамических курсоров может поддерживаться только последовательным выполнением. Оптимизатор запросов всегда формирует последовательный план выполнения для запроса, являющегося частью динамического курсора.

### Переопределение степеней параллелизма

Чтобы ограничить число процессоров для использования в параллельном плане выполнения, можно использовать параметр конфигурации сервера [max degree of parallelism](#). Для отдельного запроса и инструкций операции с индексами параметр **max degree of parallelism** можно переопределить, указав в запросе подсказку MAXDOP или параметр индекса MAXDOP. Параметр MAXDOP предоставляет улучшенное управление через отдельные запросы и операции с индексами. Например, можно использовать параметр MAXDOP для управления, увеличивая или уменьшая число процессоров, выделенных для операции с индексами в сети. Таким образом, можно сбалансировать ресурсы, используемые операцией с индексами с теми текущими пользователями. Установив параметр конфигурации сервера **max degree of parallelism** в значение 0, можно настроить SQL Server на использование всех доступных процессоров (до 64) при выполнении параллельных планов. Установив параметр конфигурации сервера MAXDOP в значение 0 для запросов и индексов, можно настроить SQL Server на использование всех доступных процессоров (до 64) при выполнении параллельных планов с данными запросами или индексами.

### Пример параллельного запроса

В нижеследующем запросе подсчитывается количество заказов, размещенных в течение указанного квартала, начиная с 1 апреля 2000, в которых хотя бы один элемент из списка заказанных товаров был получен заказчиком позже фиксированной даты. В этом запросе представлен подсчет таких заказов, сгруппированных в соответствии со срочностью каждого заказа и отсортированных в возрастающем порядке.

В этом примере используются теоретические имена таблицы и столбцов.

```
SELECT o_orderpriority, COUNT(*) AS Order_Count
FROM orders
WHERE o_orderdate >= '2000/04/01'
      AND o_orderdate < DATEADD (mm, 3, '2000/04/01')
      AND EXISTS
      (
        SELECT *
        FROM   lineitem
        WHERE  l_orderkey = o_orderkey
              AND l_commitdate < l_receiptdate
      )
GROUP BY o_orderpriority
ORDER BY o_orderpriority
```

Предположим, что нижеследующие индексы определены в таблицах **lineitem** и **orders**:

```
CREATE INDEX l_order_dates_idx
ON lineitem
(l_orderkey, l_receiptdate, l_commitdate, l_shipdate)

CREATE UNIQUE INDEX o_datkeyopr_idx
ON ORDERS
(o_orderdate, o_orderkey, o_custkey, o_orderpriority)
```

Вот один из возможных параллельных планов, созданный для запроса, показанного выше:

```
--Stream Aggregate(GROUP BY:([ORDERS].[o_orderpriority]))
      DEFINE:([Expr1005]=COUNT(*)))
--Parallelism(Gather Streams, ORDER BY:
      ([ORDERS].[o_orderpriority] ASC))
--Stream Aggregate(GROUP BY:
      ([ORDERS].[o_orderpriority]))
      DEFINE:([Expr1005]=Count(*)))
--Sort(ORDER BY:([ORDERS].[o_orderpriority] ASC))
      |--Merge Join(Left Semi Join, MERGE:
      ([ORDERS].[o_orderkey])=
      ([LINEITEM].[l_orderkey]),
      RESIDUAL:([ORDERS].[o_orderkey]=
      [LINEITEM].[l_orderkey]))
      |--Sort(ORDER BY:([ORDERS].[o_orderkey] ASC))
      |      |--Parallelism(Repartition Streams,
      PARTITION COLUMNS:
      ([ORDERS].[o_orderkey]))
      |      |--Index Seek(OBJECT:
      ([tpcd1G].[dbo].[ORDERS].[O_DATKEYOPR_IDX]),
      SEEK:([ORDERS].[o_orderdate] >=
      Apr  1 2000 12:00AM AND
      [ORDERS].[o_orderdate] <
      Jul  1 2000 12:00AM) ORDERED)
      |--Parallelism(Repartition Streams,
```

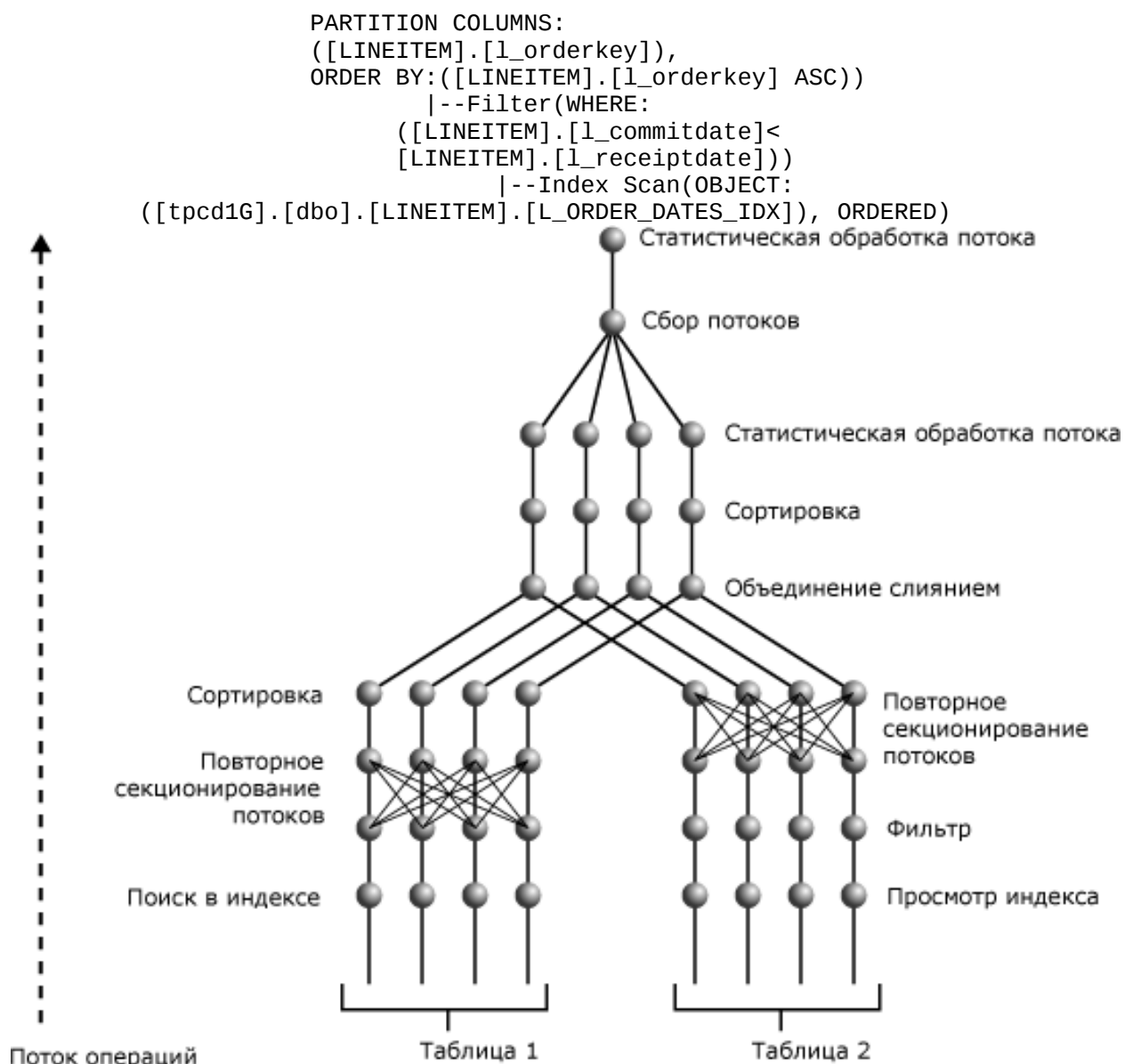


Иллюстрация демонстрирует план оптимизатора запросов, выполняемый со степенью параллелизма, равной 4, и включающий соединение двух таблиц.

Параллельный план содержит три оператора Parallelism. Оба оператора, **Index Seek** индекса **o\_datkey\_ptr** и **Index Scan** индекса **l\_order\_dates\_idx**, выполняются параллельно. В результате образуется несколько исключаяющих потоков. Это может быть определено из ближайших операторов Parallelism над операторами **Index Scan** и **Index Seek**, соответственно. Оба перераспределяют тип обмена. То есть они всего лишь перегруппируют данные между потоками и выдают в результате столько же потоков на выходе, сколько их было на входе. Количество потоков равно степени параллелизма.

Оператор **Parallelism** над оператором **Index Scan** **l\_order\_dates\_idx** перераспределяет свои входные потоки с использованием значения **L\_ORDERKEY** в качестве ключа. В этом случае те же значения **L\_ORDERKEY** выдаются в том же выходном потоке. Одновременно в потоках выхода поддерживается порядок в столбце **L\_ORDERKEY** для соответствия входному требованию оператора **Merge Join**.

Оператор **Parallelism** над оператором **Index Seek** перераспределяет свои входные потоки с использованием значения **O\_ORDERKEY**. Поскольку его входные данные не сортируются по значениям столбца **O\_ORDERKEY**, а он является столбцом соединения в операторе **Merge Join**, оператор **Sort** между операторами **Parallelism** и **Merge Join** обеспечивает сортировку входных данных для оператора **Merge Join** по столбцам соединения. Оператор **Sort**, как и оператор **Merge Join**, выполняется параллельно.

Первый оператор **Parallelism** соединяет результаты из нескольких потоков в один. Результаты частичной статистической обработки, выполняемой оператором **Stream Aggregate** под оператором **Parallelism**, затем собираются в единое значение **SUM** для каждого отдельного значения **O\_ORDERPRIORITY** в операторе **Stream Aggregate** над оператором **Parallelism**. Поскольку этот план состоит из двух сегментов обмена со степенью параллелизма, равной 4, он использует восемь потоков.

### *Параллельные операции с индексами*

Планы запросов, построенные для операций создания или перестроения индекса либо удаления кластеризованного индекса, поддерживают возможность параллельной, многопоточной обработки на многопроцессорных компьютерах.

#### **Примечание**

Параллельные операции с индексами доступны только в выпуске SQL Server 2008 Enterprise Edition.

В SQL Server используются одни и те же алгоритмы определения степени параллелизма (общего числа запускаемых отдельных потоков) как для операций с индексами, так и для других запросов. Максимальная степень параллелизма для операции с индексом определяется параметром конфигурации сервера [max degree of parallelism](#).

Значение **max degree of parallelism** можно переопределять для отдельных операций с индексами путем настройки параметра индекса **MAXDOP** в инструкциях **CREATE INDEX**, **ALTER INDEX**, **DROP INDEX** и **ALTER TABLE**.

Когда компонент Database Engine строит план выполнения индекса, количество параллельных операций устанавливается на наиболее низкое из перечисленных ниже значений.

- Число микропроцессоров (ЦП) в компьютере.
- Число, указанное в качестве параметра конфигурации сервера **max degree of parallelism**.
- Число ЦП, не превышающих порог загруженности рабочими потоками SQL Server.

Например, на компьютере с восемью ЦП, на котором значение **max degree of parallelism** равно 6, для операции с индексом создается не более шести параллельных потоков. Если пять ЦП на компьютере загружены рабочими потоками SQL Server сверх порогового значения при построении плана выполнения индекса, в этом плане задаются только три параллельных потока.

Главные фазы параллельных операций с индексами таковы.

- Координирующий поток быстро и случайным образом просматривает таблицу

для оценки распределения ключей индекса. Координирующий поток устанавливает ключевые границы, образующие число диапазонов ключей, равное степени параллелизма, где каждый диапазон по оценке должен покрывать примерно одинаковое число строк. Например, если в таблице 4 миллиона строк, а степень параллелизма равна 4, координирующий поток определит ключевые значения, ограничивающие четыре набора строк по миллиону каждый. Если для использования всех ЦП невозможно установить достаточное число диапазонов ключей, степень параллелизма соответствующим образом снижается.

- Координирующий поток запускает число потоков, равное степени параллелизма операций, и ожидает завершения работы этих потоков. Каждый из потоков просматривает базовую таблицу, используя фильтр, получающий только строки с ключевыми значениями в пределах диапазона, назначенного данному потоку. Каждый поток выполняет построение индексной структуры для строк в своем диапазоне ключей. В случае секционированного индекса каждый из потоков выполняет построение указанного числа секций. Одни и те же секции между несколькими потоками не разделяются. Дополнительные сведения о построении индекса см. в разделе [База данных tempdb и создание индекса](#).

- После завершения работы всех параллельных потоков координирующий поток связывает компоненты индекса в единый индекс. Эта фаза применяется только для автономных операций с индексами.

В отдельных инструкциях CREATE TABLE или ALTER TABLE могут содержаться несколько ограничений, требующих создания индекса. Такие множественные операции по созданию индекса выполняются последовательно, хотя каждая из них может быть параллельной операцией на многопроцессорном компьютере.

## Архитектура распределенных запросов

Microsoft SQL Server поддерживает два метода обращения к разнородным источникам данных OLE DB в инструкциях языка Transact-SQL.

- Имена связанных серверов  
Системные хранимые процедуры **sp\_addlinkedserver** и **sp\_addlinkedsrvlogin** используются для задания серверного имени источнику данных OLE DB. К объектам на этих связанных серверах можно обращаться в инструкциях языка Transact-SQL по четырехкомпонентным именам. Например, если имя связанного сервера **DeptSQLSrvr** определено для другого экземпляра SQL Server, для обращения к таблице на таком сервере используется следующая инструкция:

```
SELECT JobTitle, HireDate
FROM DeptSQLSrvr.AdventureWorks2008R2.HumanResources.Employee;
```

Имя связанного сервера можно также указать в инструкции OPENQUERY для открытия набора строк из источника данных OLE DB. К этому набору строк можно обращаться в инструкциях языка Transact-SQL так же, как и к таблице:

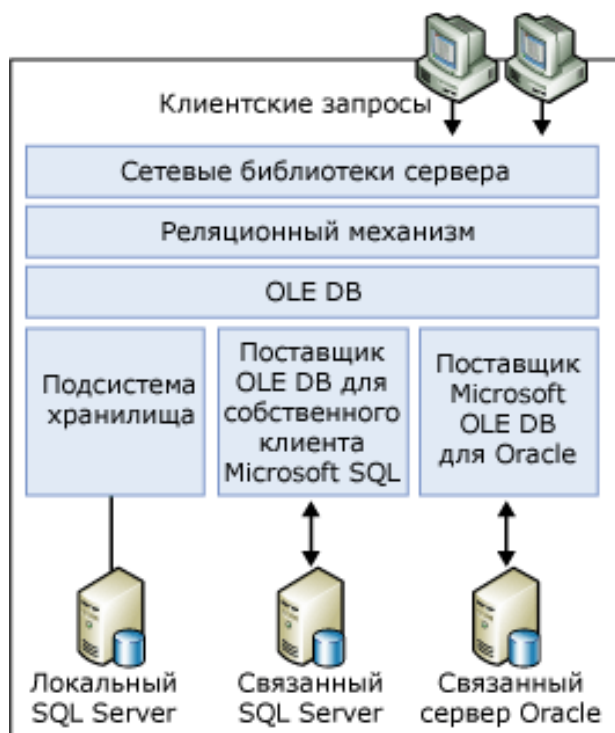
- Имена нерегламентированных соединителей  
Для нечастых обращений к источнику данных используются функции OPENROWSET или OPENDATASOURCE, которыми задаются данные, необходимые для подключения к связанному серверу. Затем можно обращаться к набору строк в инструкциях языка Transact-SQL тем же путем, что и к таблице:

```
SELECT *
FROM OPENROWSET('Microsoft.Jet.OLEDB.4.0',
```



```
'c:\MSOffice\Access\Samples\Northwind.mdb';'Admin';";  
Employees);
```

В SQL Server для коммуникации между реляционным механизмом и подсистемой хранилища используется технология OLE DB. Реляционный механизм разбивает каждую инструкцию языка Transact-SQL на последовательные операции над простыми наборами строк OLE DB, открываемые подсистемой хранилища из базовых таблиц. Это означает, что реляционный механизм может также открывать простые наборы строк OLE DB на любом источнике данных OLE DB.



В реляционном механизме используется прикладной программный интерфейс (API) OLE DB для открытия наборов строк на связанных серверах, выборки строк и управления транзакциями.

Для каждого источника данных OLE DB, доступ к которому осуществляется как к связанному серверу, на сервере с запущенной службой SQL Server должен быть поставщик OLE DB. Набор операций языка Transact-SQL, которые можно использовать с конкретным источником данных OLE DB, зависит от возможностей поставщика OLE DB. Дополнительные сведения см. в разделе [Справочник по поставщику OLE DB для распределенных запросов](#).

Для каждого экземпляра SQL Server члены предопределенной роли сервера **sysadmin** могут включать или отключать использование нерегламентированных имен соединителей для поставщика OLE DB с помощью свойства SQL Server **DisallowAdhocAccess**. Если нерегламентированный доступ включен, любой пользователь, зарегистрированный на данном экземпляре, может выполнять инструкции SQL, содержащие имена нерегламентированных соединителей, обращающиеся к любым источникам данных в сети, доступ к которым возможен посредством данного поставщика OLE DB. Для управления доступом к источникам данных члены роли **sysadmin** могут отключить нерегламентированный доступ к определенному



поставщику OLE DB, ограничивая таким образом пользователей доступом лишь к тем источникам данных, обращение к которым производится по именам связанных серверов, определенным администраторами. По умолчанию нерегламентированный доступ включен для поставщика OLE DB SQL Server и отключен для всех остальных поставщиков OLE DB.

Распределенные запросы могут давать пользователям доступ к другому источнику данных (например, файлам, нереляционным источникам данных типа службы Active Directory и т. д.) с помощью контекста безопасности учетной записи Microsoft, под которой запущена служба SQL Server. Служба SQL Server может правильно олицетворять имена входа Windows, однако для имен входа SQL Server это невозможно. Это потенциально может открыть пользователю распределенного запроса доступ к другому источнику данных, для которого у него нет разрешения, но у учетной записи, под которой запущена служба SQL Server, такое разрешение есть. Для указания конкретных имен входа, которым будет разрешен доступ к соответствующему связанному серверу, используется процедура `sp_addlinkedserverlogin`. Для нерегламентированных имен такой контроль недоступен, поэтому следует проявлять осторожность при включении нерегламентированного доступа к поставщику OLE DB.

При возможности SQL Server принудительно отправляет реляционные операции — соединения, ограничения, проекции, сортировки и группировки по операциям — к источнику данных OLE DB. По умолчанию SQL Server не просматривает базовую таблицу и не выполняет реляционных операций самостоятельно. SQL Server запрашивает поставщика OLE DB, чтобы определить уровень поддерживаемой им грамматики SQL, и на основе этих данных направляет поставщику максимально возможное число реляционных операций. Дополнительные сведения см. в разделе [Требования к диалекту SQL для поставщиков OLE DB](#).

SQL Server указывает поставщику OLE DB механизм возвращения статистики распределения ключевых значений в пределах источника данных OLE DB. Это позволяет оптимизатору запросов SQL Server лучше анализировать закономерность в соответствии данных из источника данных требованиям каждой инструкции SQL, что увеличивает способность оптимизатора создавать оптимальные планы выполнения. Дополнительные сведения см. в разделе [Требования к статистике распределения для поставщиков OLE DB](#).

## Улучшенные возможности обработки запросов для секционированных таблиц и индексов

SQL Server 2008 повышает производительность обработки запросов к секционированным таблицам для многих параллельных планов, изменяет способ представления параллельных и последовательных планов и улучшает информацию о секционировании, содержащуюся в планах времени компиляции и в планах времени выполнения. В этом разделе описываются названные улучшения, содержится справочник об интерпретации планов выполнения запросов секционированных таблиц и индексов и дополнительные сведения об улучшении производительности запросов к

секционированным объектам.

#### Примечание

Секционированные таблицы и индексы поддерживаются только в выпусках SQL Server Enterprise Edition, Developer Edition и Evaluation Edition.

## Новая операция поиска, учитывающая секционирование

В SQL Server 2008 внутреннее представление секционированной таблицы изменено таким образом, что таблица представляется обработчику запросов как индекс по нескольким столбцам с PartitionID в качестве начального столбца. PartitionID представляет собой скрытый внутренний вычисляемый столбец для представления идентификатора секции, содержащей определенную строку. Например, предположим, что таблица T, определенная как T(a, b, c), секционирована по столбцу a и содержит кластеризованный индекс по столбцу b. В SQL Server 2008 эта секционированная таблица обрабатывается внутри как несекционированная таблица со схемой T(PartitionID, a, b, c) и кластеризованным индексом по составному ключу (PartitionID, b). Это позволяет оптимизатору запросов выполнять операции поиска, основанные на PartitionID, по любой секционированной таблице или индексу.

Устранение секций теперь осуществляется в этой операции поиска.

Кроме того, оптимизатор запросов расширен таким образом, что операция поиска или просмотра с одним условием может быть выполнена по PartitionID (в качестве логического начального столбца) и, возможно, по другим столбцам ключа индекса, а затем может быть выполнен поиск второго уровня с другим условием по одному дополнительному столбцу или более для каждого уникального значения, удовлетворяющего операции поиска первого уровня. Операция, называемая просмотр с пропуском, позволяет оптимизатору запросов выполнять операцию поиска или просмотра на основе одного условия для определения секций, к которым осуществляется доступ, и операцию поиска индекса второго уровня с помощью этого оператора для выборки строк из этих секций, удовлетворяющих другому условию. Например, рассмотрим следующий запрос.


```
SELECT * FROM T WHERE a < 10 and b = 2;
```

В данном примере, предположим, таблица T, определенная как T(a, b, c), секционирована по столбцу a и содержит кластеризованный индекс по столбцу b. Границы секции для таблицы T определены следующей функцией секционирования:

```
CREATE PARTITION FUNCTION myRangePF1 (int) AS RANGE LEFT FOR VALUES (3, 7, 10);
```

Для разрешения запроса обработчик запросов выполняет операцию поиска первого уровня для нахождения каждой секции, содержащей строки, удовлетворяющие условию T.a < 10. Это позволяет выявить секции, к которым необходимо получить доступ. В каждой выявленной секции обработчик выполняет поиск второго уровня по кластеризованному индексу по столбцу b для нахождения строк, удовлетворяющих условию T.b = 2 и T.a < 10.

На следующем рисунке изображено логическое представление операции просмотра с пропуском. На нем изображена таблица T с данными в столбцах a и b. Секции пронумерованы от 1 до 4, а границы секций показаны вертикальными штриховыми линиями. Операция поиска первого уровня для секций (на иллюстрации не показана) определила, что секции 1, 2 и 3 удовлетворяют условию поиска, предполагаемого секционированием, определенным для таблицы и предиката по столбцу a, то есть  $T.a < 10$ . Путь, пройденный частью операции просмотра с пропуском, поиском второго уровня, изображен изогнутой линией. Фактически операция просмотра с пропуском выполняет поиск строк, удовлетворяющих условию  $b = 2$  в каждой из этих секций. Общие затраты на выполнение операции просмотра с пропуском соответствуют трем отдельным поискам по индексу.



Идентификатор секции	1	1	1	2	2	2	2	3	3	3	4	4	4
Столбец b	1	2	4	1	2	4	5	2	2	3	1	1	4
Столбец a	1	2	3	4	5	6	7	8	8	10	11	11	12

## Отображение сведений о секционировании в планах выполнения запросов

Планы выполнения запросов в секционированных таблицах и индексах могут быть исследованы с помощью инструкций SET языка Transact-SQL, SET SHOWPLAN\_XML или SET STATISTICS XML, либо с помощью графического представления плана выполнения в среде Средства SQL Server Management Studio. Например, план выполнения времени компиляции можно отобразить, щелкнув Показать предполагаемый план выполнения на панели инструментов редактора запросов, а план времени выполнения — щелкнув Включить действительный план выполнения.

С помощью этих средств можно получить следующую информацию:

- операции, такие как проверка, поиск, вставка, обновление, слияние и удаление, которые осуществляют доступ к секционированным таблицам и индексам;
- секции, к которым запрос получает доступ — например, в планах времени выполнения приведено общее число секций, к которым получен доступ, и диапазоны смежных секций, к которым получен доступ;
- когда операция просмотра с пропуском используется в операции поиска или просмотра для получения данных из одной секции или более.

Дополнительные сведения об отображении планов выполнения см. в разделе [Инструкции по планам выполнения](#).

### Улучшенные возможности информации о секции

SQL Server 2008 содержит расширенные сведения о секционировании как для планов

времени компиляции, так и для планов времени выполнения. Планы выполнения теперь содержат следующую информацию.

- Дополнительный атрибут **Partitioned** указывает, что оператор, например поиска, проверки, вставки, обновления, слияния или удаления, выполняется в отношении секционированной таблицы.
- Новый элемент **SeekPredicateNew** с подэлементом **SeekKeys**, содержащим PartitionID в качестве начального столбца ключа индекса и условия фильтра, определяющие операции поиска по диапазону в PartitionID. Наличие двух подэлементов **SeekKeys** указывает на то, что в отношении PartitionID используется операция просмотра с пропуском.
- Сводные данные об общем числе секций, к которым получен доступ. Эта информация доступна только в планах времени выполнения.

Для демонстрации отображения этой информации как в графическом плане выполнения, так и в отчете инструкции XML Showplan рассмотрим следующий запрос по таблице секционирования fact\_sales. Этот запрос обновляет данные в двух секциях.

```
UPDATE fact_sales
```

```
SET quantity = quantity * 2
```

```
WHERE date_id BETWEEN 20080802 AND 20080902;
```

На следующем рисунке показаны свойства оператора **Clustered Index Seek** в плане выполнения времени компиляции для этого запроса. Определение таблицы fact\_sales и определение секции см. в подразделе «Пример» в этом разделе.

Clustered Index Seek	
Просмотр определенного диапазона строк кластеризованного индекса.	
Физическая операция	Clustered Index Seek
Логическая операция	Clustered Index Seek
Фактическое количество строк	967430
Предполагаемая стоимость опе...	102,021
Предполагаемая стоимость про...	1,06021
Предполагаемая стоимость...	103,081 (5%)
Предполагаемая стоимость...	103,081
Предполагаемое количество строк	963681
Предполагаемый размер строки	23Б
Фактическое число повторных привязок	0
Фактическое число перезапусков опе...	0
Секционировано	True
Фактическое количество секций	2
Отсортировано	True
Идентификатор узла	3
<b>Объект</b>	
[db_sales_test].[dbo].[fact_sales].[ci]	
<b>Список выходных столбцов</b>	
PtnId1000, Uniq1003, [db_sales_test].[dbo].[fact_sales].date_id, [db_sales_test].[dbo].[fact_sales].quantity	
<b>Предикаты поиска</b>	
Seek Keys[1]: Start: PtnId1000 >= Scalar Operator (RangePartitionNew([@2],(1),(20080801),(20080901),(20081001),(20081101),(20081201),(20090101))), End: PtnId1000 <= Scalar Operator(RangePartitionNew([@3],(1),(20080801),(20080901),(20081001),(20081101),(20081201),(20090101))), Seek Keys[2]: Start: [db_sales_test].[dbo].[fact_sales].date_id >= Scalar Operator([@2]), End: [db_sales_test].[dbo].[fact_sales].date_id <= Scalar Operator([@3])	

## Атрибут Partitioned

Когда оператор, такой как **Index Seek**, выполняется по секционированной таблице или индексу, в планах времени компиляции и времени выполнения появляется атрибут **Partitioned** со значением True (1). Этот атрибут не отображается, если его значение установлено как False (0).

Атрибут **Partitioned** может встречаться в следующих физических и логических операторах:

- Table Scan
- Index Scan
- Index Seek
- Insert
- Update

- **Delete**
- **Merge**

Как показано на предыдущей иллюстрации, этот атрибут отображается в свойствах оператора, в котором он определен. В отчете инструкции XML Showplan этот атрибут появляется как **Partitioned="1"** в узле **RelOp** оператора, в котором он определен.

### Предикат New Seek

В отчете инструкции XML Showplan элемент **SeekPredicateNew** появляется в операторе, в котором он определен. Он может содержать до двух экземпляров вложенного элемента **SeekKeys**. Первый элемент **SeekKeys** определяет операцию поиска первого уровня на уровне идентификатора секции логического индекса. То есть эта операция поиска определяет секции, к которым должен быть осуществлен доступ для удовлетворения условий запроса. Второй элемент **SeekKeys** определяет часть операции просмотра с пропуском, поиск второго уровня, который производится в каждой секции, определенной поиском первого уровня.

### Сводные данные по секциям

В планах времени выполнения сводка по секциям содержит данные о числе секций, к которым осуществлен доступ, и фактический перечень секций, к которым осуществлен доступ. С помощью этих данных можно проверить, к правильным ли секциям обращается запрос и исключены ли из рассмотрения остальные секции.

Предоставляется следующая информация: **Actual Partition Count** и **Partitions Accessed**.

**Actual Partition Count** — это общее число секций, к которым запрос получает доступ.

**Partitions Accessed** в отчете инструкции XML Showplan — это сводные данные по секциям, которые появляются в новом элементе **RuntimePartitionSummary** в узле **RelOp** оператора, в котором он определен. В следующем примере показано содержимое элемента **RuntimePartitionSummary**, указывающее, что всего получен доступ к двум секциям (секции 2 и 3).

```
<RunTimePartitionSummary>
  <PartitionsAccessed PartitionCount="2" >
    <PartitionRange Start="2" End="3" />
  </PartitionsAccessed>
</RunTimePartitionSummary>
```

### Отображение сведений о секционировании с помощью других методов Showplan

Методы Showplan SHOWPLAN\_ALL, SHOWPLAN\_TEXT и STATISTICS PROFILE не формируют сведения о секционировании, описанные в этом разделе, за следующим исключением. Как часть предиката **SEEK**, секции, к которым необходимо получить доступ, обозначаются предикатом по диапазону в вычисляемом столбце, представляющем идентификатор секций. В следующем примере показан

предикат **SEEK** для оператора **Clustered Index Seek**. К секциям 2 и 3 происходит обращение, и оператор поиска производит фильтрацию по строкам, удовлетворяющим условию `date_id BETWEEN 20080802 AND 20080902`.

```
--Clustered Index Seek(OBJECT:([db_sales_test].[dbo].[fact_sales].[ci]),
    SEEK:([PtnId1000] >= (2) AND [PtnId1000] <= (3)
        AND [db_sales_test].[dbo].[fact_sales].[date_id] >= (20080802)
        AND [db_sales_test].[dbo].[fact_sales].[date_id] <= (20080902)
    ORDERED FORWARD)
```

## Интерпретация планов выполнения для секционированной кучи

В SQL Server 2008 секционированная куча обрабатывается как логический индекс по идентификатору секции. Устранение секций на секционированной куче представлено в плане выполнения в виде оператора [Table Scan](#) с предикатом **SEEK** по идентификатору секции. Следующий пример отображает сведения Showplan:

```
-- Table Scan (OBJECT: ([db].[dbo].[T]), SEEK: ([PtnId1001]=[Expr1011]) ORDERED
FORWARD)
```

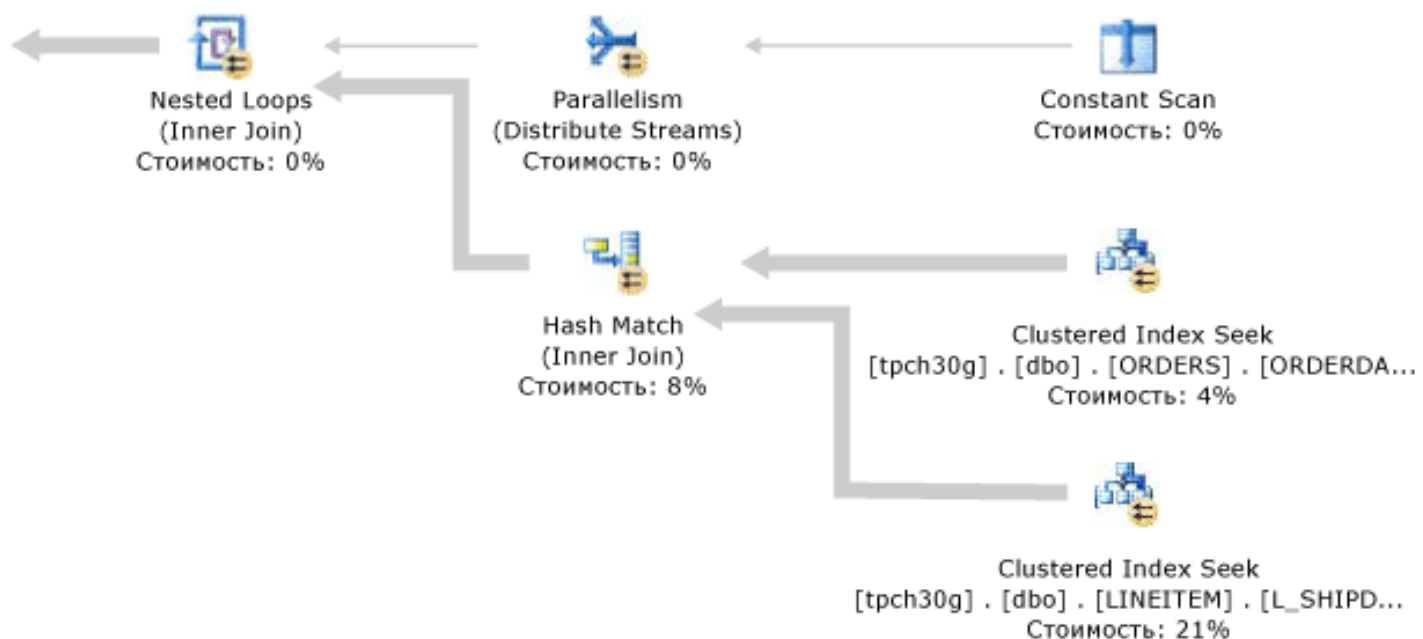
## Интерпретация планов выполнения для выровненных соединений

Выровненное соединение может возникать, когда две таблицы секционированы с использованием одной и той же функции или эквивалентных функций секционирования и столбцы секционирования из обеих сторон соединения указываются в условии соединения запроса. Оптимизатор запросов может сформировать план, в котором секции каждой таблицы, имеющие равные идентификаторы, соединяются отдельно. Выровненные соединения могут выполняться быстрее, чем невыровненные, поскольку требуют меньшего объема памяти и времени обработки. Оптимизатор выбирает невыровненный план или выровненный план исходя из расчета затрат.

В выровненных планах соединение **Nested Loops** считывает одну или более секций для соединяемых таблиц или индексов с внутренней стороны. Цифры в операторах **Constant Scan** представляют номера секций.

Если для секционированных таблиц или индексов формируются параллельные планы для выровненных соединений, то между операторами соединения **Constant Scan** и **Nested Loops** появляется оператор [Parallelism](#). В этом случае каждый из нескольких потоков на внешней стороне соединения считывает и работает на разных секциях.

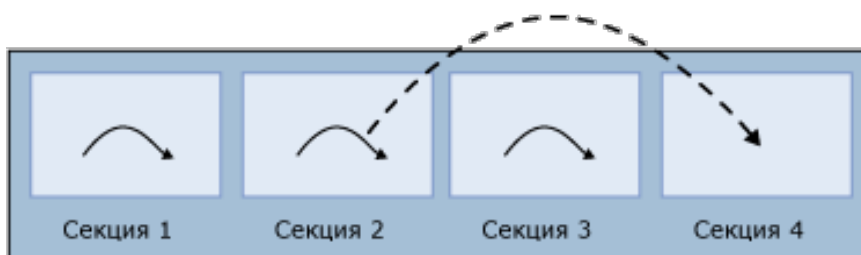
Следующий рисунок демонстрирует план параллельных запросов для выровненных соединений.



## Стратегия выполнения параллельных запросов для секционированных объектов

Обработчик запросов использует стратегию параллельного выполнения для запросов, производящих выборку из секционированных объектов. В рамках стратегии выполнения обработчик запросов определяет секции таблицы, необходимые для запроса, и долю потоков, которую следует выделить для каждой секции. В большинстве случаев обработчик запросов выделяет равное или почти равное количество потоков для каждой секции, а затем выполняет запрос параллельно на всех секциях. В следующих параграфах более подробно разъясняется выделение потоков.

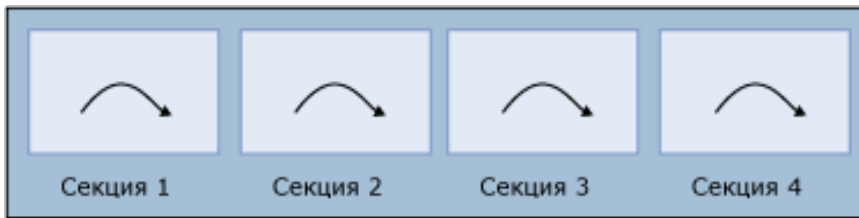
Если число потоков меньше числа секций, обработчик запросов присваивает по одному потоку для каждой отдельной секции, изначально оставляя несколько секций без присвоенных потоков. Когда поток завершает работу с секцией, обработчик запросов присваивает его следующей секции. Это продолжается до тех пор, пока у каждой секции не будет по одному потоку. Это единственный случай, когда обработчик запросов перераспределяет потоки к другим секциям.



Если число потоков равно числу секций, обработчик запросов присваивает каждой секции по одному потоку. После того как поток заканчивает работу, он не



перераспределяется к другой секции.



Если число потоков больше числа секций, обработчик запросов присваивает каждой секции одинаковое число потоков. Если число потоков не кратно числу секций, обработчик запросов выделяет по одному дополнительному потоку для некоторых секций, чтобы были использованы все доступные потоки. Следует заметить, что если существует только одна секция, ей будут присвоены все потоки. На рисунке, приведенном ниже, показаны четыре секции и 14 потоков. Каждой секции присвоено по 3 потока, у двух секций есть дополнительные потоки; всего присвоено 14 потоков. После того как поток заканчивает работу, он не перераспределяется к другой секции.



Хотя в приведенных выше примерах показывается достаточно прямолинейный способ распределения потоков, реальная стратегия более сложна; она учитывает другие факторы, которые возникают при выполнении запроса. Например, если таблица секционирована и имеет кластеризованный индекс на столбце A, а в запросе используется предложение предиката WHERE A IN (13, 17, 25), обработчик запросов присвоит один или несколько потоков каждому из трех искомых значений из значений поиска (A=13, A=17 и A=25) (а не каждой секции таблицы). Запрос необходимо выполнить только в секциях, содержащих эти значения; если все предикаты поиска будут расположены в одной секции таблицы, все потоки будут присвоены этой секции.

Другой пример: предположим, что таблица имеет четыре секции для столбца A с граничными точками (10, 20, 30), индекс на столбце B, а в запросе содержится предикат WHERE B IN (50, 100, 150). Поскольку секции таблицы основаны на значениях A, значения столбца B могут появляться во всех секциях таблицы. Поэтому обработчик запросов будет искать каждое из этих трех значений столбца B (50, 100, 150) в каждой из четырех секций таблицы. Обработчик запросов распределит потоки пропорционально, чтобы эти 12 просмотров запроса могли выполняться параллельно.

Секции таблицы основаны на столбце A	Операции поиска для столбца B в каждой секции таблицы
Секция таблицы 1: A < 10	B = 50, B = 100, B = 150

Секция таблицы 2: A >= 10 AND A < 20	B = 50, B = 100, B = 150
Секция таблицы 3: A >= 20 AND A < 30	B = 50, B = 100, B = 150
Секция таблицы 4: A >= 30	B = 50, B = 100, B = 150

## Советы и рекомендации

Для увеличения производительности запросов, обращающихся к большому количеству данных из больших секционированных таблиц и индексов, предлагаются следующие рекомендации.

- Распределяйте каждую секцию по нескольким дискам.
- Чтобы снизить затраты на ввод-вывод, по возможности используйте сервер с достаточным объемом основной памяти, вмещающей секции, требующие частого доступа, или все секции.
- Если данные, по которым выполняется запрос, не помещаются в памяти, рекомендуется выполнить сжатие таблиц и индексов. Это позволит снизить затраты на ввод-вывод.
- Чтобы в полной мере реализовать возможности параллельной обработки запросов, используйте сервер с быстрыми процессорами и как можно большим числом процессорных ядер.
- Обеспечьте достаточную пропускную способность контроллера ввода-вывода для сервера.
- Чтобы в полной мере реализовать возможности оптимизированного просмотра сбалансированного дерева, создайте кластеризованный индекс по каждой большой секционированной таблице.
- При массовой загрузке данных в секционированные таблицы следуйте рекомендациям технического документа по [массовой загрузке данных в секционированные таблицы](#).

## Пример

В следующем примере показано создание тестовой базы данных, состоящей из одной таблицы с семью секциями. Чтобы при выполнении запросов в этом примере просматривать сведения о секционировании в планах времени компиляции и времени выполнения, следует воспользоваться инструментами, описанными ранее. Внесение оборудования выполнения данного примера может занять несколько минут. Перед выполнением этого примера следует убедиться, что на диске 1,5 ГБ свободного места.

```
USE master;  
GO
```

---

```
IF DB_ID (N'db_sales_test') IS NOT NULL
    DROP DATABASE db_sales_test;
GO
CREATE DATABASE db_sales_test;
GO
USE db_sales_test;
GO
CREATE PARTITION FUNCTION [pf_range_fact](int) AS RANGE RIGHT FOR VALUES
(20080801, 20080901, 20081001, 20081101, 20081201, 20090101);
GO
CREATE PARTITION SCHEME [ps_fact_sales] AS PARTITION [pf_range_fact]
ALL TO ([PRIMARY]);
GO
CREATE TABLE fact_sales(date_id int, product_id int, store_id int,
    quantity int, unit_price numeric(7,2), other_data char(1000))
ON ps_fact_sales(date_id);
GO
CREATE CLUSTERED INDEX ci ON fact_sales(date_id);
GO
PRINT 'Loading...';
SET NOCOUNT ON;
DECLARE @i int;
SET @i = 1;
WHILE (@i<1000000)
BEGIN
    INSERT INTO fact_sales VALUES(20080800 + (@i%30) + 1, @i%10000, @i%200,
RAND() * 25, (@i%3) + 1, '');
    SET @i += 1;
END;
GO
DECLARE @i int;
SET @i = 1;
WHILE (@i<10000)
BEGIN
    INSERT INTO fact_sales VALUES(20080900 + (@i%30) + 1, @i%10000, @i%200,
RAND() * 25, (@i%3) + 1, '');
    SET @i += 1;
END;
PRINT 'Done.';
GO
-- Two-partition query.
SET STATISTICS XML ON;
GO
SELECT date_id, SUM(quantity*unit_price) AS total_price
FROM fact_sales
WHERE date_id BETWEEN 20080802 AND 20080902
GROUP BY date_id ;
GO
SET STATISTICS XML OFF;
GO
-- Single-partition query.
SET STATISTICS XML ON;
GO
SELECT date_id, SUM(quantity*unit_price) AS total_price
FROM fact_sales
WHERE date_id BETWEEN 20080801 AND 20080831
GROUP BY date_id;
GO
SET STATISTICS XML OFF;
```

GO

## Архитектура управления памятью

В этом разделе описывается архитектура управления памятью компонента SQL Server Database Engine.

### Архитектура оперативной памяти

SQL Server по мере необходимости динамически получает и освобождает оперативную память. Обычно администратору не требуется указывать, сколько памяти необходимо выделить для SQL Server, хотя эта возможность по-прежнему существует и в некоторых случаях необходима.

SQL Server поддерживает расширения AWE, позволяя использовать более 4 ГБ физической памяти на 32-разрядных версиях операционной системы Microsoft Windows. Поддерживается до 64 ГБ физической памяти. Экземпляры SQL Server, запущенные на Microsoft Windows 2000, используют статическое распределение памяти AWE, в то время как экземпляры, запущенные на Microsoft Windows Server 2003, — динамическое.

#### Примечание

Поддержка AWE доступна только в выпусках SQL Server Enterprise Edition, Standard Edition и Developer Edition для 32-разрядных операционных систем. Преимущества расширенной памяти AWE недоступны для служб Службы Analysis Services. Если объем доступной физической памяти меньше, чем пространство виртуальных адресов пользовательского режима, то AWE включить нельзя.

Одной из главных задач проектирования любой программной системы для баз данных является минимизация операций дискового ввода-вывода, поскольку чтение и запись на диск являются наиболее ресурсоемкими операциями. Серверы SQL Server создают в памяти буферные пулы для удержания страниц, считываемых из базы данных. Большой объем кода SQL Server предназначен для минимизации числа физических операций считывания и записи между диском и буферным пулом. SQL Server пытается найти оптимальное соотношение при достижении двух целей:

- предотвращения роста буферного пула до размера, при котором вся система будет испытывать нехватку оперативной памяти;
- минимизации числа физических операций ввода-вывода в базе данных путем увеличения размера буферного пула.

Дополнительные сведения см. в разделе [Управление буферами](#).

По умолчанию все выпуски SQL Server 2005 динамически управляют памятью для каждого экземпляра. Существуют отличия в том, как SQL Server управляет AWE-памятью в Windows 2000 и в более поздних версиях этой операционной системы.

#### Примечание

В сильно загруженной системе некоторые масштабные запросы, для выполнения которых требуется большой объем оперативной памяти, не могут получить минимально необходимого им объема и в результате завершаются ошибкой, когда истекает время ожидания ресурса памяти. Для решения этой проблемы следует увеличить значение параметра [query wait](#). В случае параллельных запросов следует рассмотреть возможность уменьшения значения параметра [max degree of parallelism](#).

#### Примечание

В сильно загруженной системе с ограниченными ресурсами памяти запросы, содержащие соединение слиянием, сортировку и построение битовой карты в плане запроса, могут удалить битовую карту, если запрос не получил минимально необходимого для ее сохранения объема оперативной памяти. Это может повлиять на производительность запроса, и, если процесс сортировки не помещается в памяти, приводит к повышению интенсивности использования рабочих таблиц в базе данных **tempdb**, вызывая тем самым ее рост. Для решения этой проблемы необходимо увеличить объем физической памяти или настроить запросы так, чтобы они использовали другие, более быстрые планы запроса. Дополнительные сведения о настройке см. в разделах [Оптимизация производительности базы данных tempdb](#) и [Как настроить базу данных](#).

## Выделение SQL Server максимального объема памяти

С помощью AWE и разрешения **Закрепление страниц в памяти** компоненту SQL Server Database Engine можно выделять следующие объемы памяти.

	32-разрядная версия	64-разрядная версия
Обычная память	<p>Все выпуски SQL Server: до достижения предела обработки виртуального адресного пространства —</p> <ul style="list-style-type: none"> <li>• 2 ГБ</li> <li>• 3 ГБ с параметром загрузки/3gb 1</li> <li>• 4 ГБ на WOW642</li> </ul>	<p>Все выпуски SQL Server: до достижения предела обработки виртуального адресного пространства —</p> <ul style="list-style-type: none"> <li>• 7 ТБ для архитектуры IA64</li> <li>• 8 ТБ для архитектуры x64</li> </ul> <p><b>Примечание</b> Для Windows Server 2003 существует ограничение 512 ГБ, а для Windows Server 2003 с пакетом обновления 1 (SP1) ограничение составляет 1 ТБ. Если Windows поддерживает дополнительную память, SQL Server может достичь приведенных в списке пределов.</p>
Механизм AWE (позволяет SQL Server выходить за пределы)	Выпуски SQL Server Standard, Enterprise и Developer: буферный пул,	Неприменимо3.

обработки виртуального адресного пространства на 32-разрядной платформе).	достаточный для доступа к 64 ГБ памяти.	
Разрешение операционной системы закреплять страницы в памяти (разрешает блокировать физическую память, предотвращает разбиение заблокированной памяти на страницы) <sup>4</sup> .	Выпуски SQL Server Standard, Enterprise и Developer: требуется для процесса SQL Server для использования механизма AWE. Память, распределенная с использованием механизма AWE, не может быть выгружена. Предоставление данного разрешения без включения AWE не оказывает влияния на сервер.	Выпуски SQL Server Enterprise и Developer: рекомендуется, чтобы избежать выгрузки страниц операционной системой. В зависимости от рабочей нагрузки может улучшить производительность. Объем доступной памяти аналогичен объему в случае обычной памяти.

1 /3gb — это параметр загрузки операционной системы. Дополнительные сведения см. в [библиотеке MSDN](#).

2 WOW (Windows on Windows 64) — режим, в котором 32-разрядная версия SQL Server запускается в 64-разрядной операционной системе. Дополнительные сведения см. в [библиотеке MSDN](#).

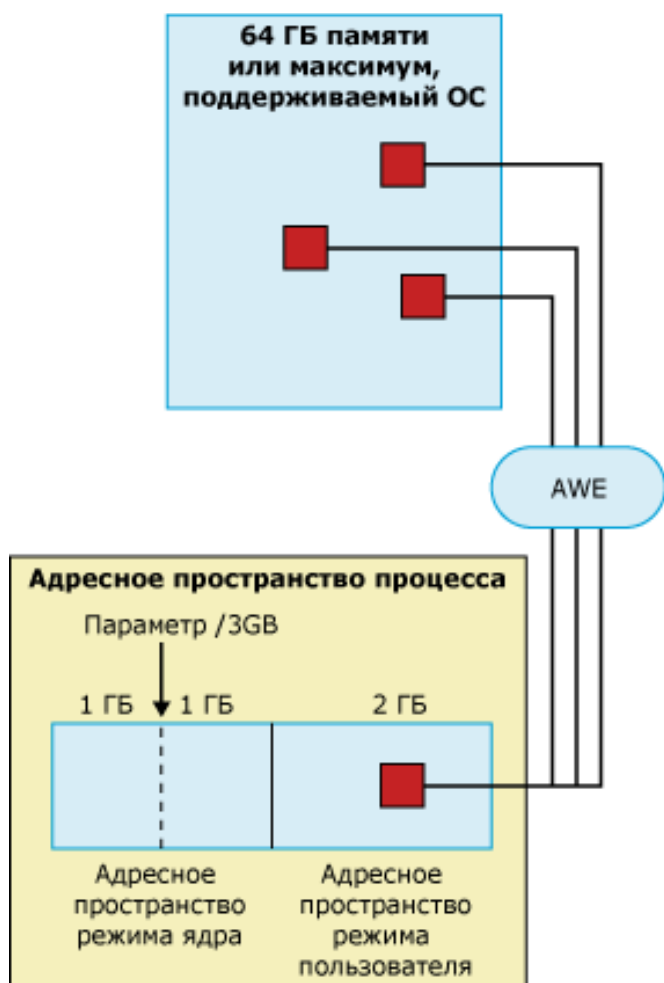
3 Обратите внимание, что параметр [awe\\_enabled](#) функции **sp\_configure** в 64-разрядной системе SQL Server присутствует, но не учитывается. Он будет исключен в будущих 64-разрядных версиях или пакетах обновления SQL Server.

4 В случае выдачи разрешения на закрепление страниц в памяти (либо в 32-разрядной системе для поддержки AWE-памяти, либо в 64-битной системе), рекомендуется выделить максимальный объем серверной памяти. Дополнительные сведения см. в разделе [Параметры памяти сервера](#).

## Адресное пространство процесса

Все 32-разрядные приложения имеют адресное пространство процесса размером 4 ГБ (то есть для 32-разрядного приложения может быть выделено не более 4 ГБ памяти). Операционная система Microsoft Windows предоставляет приложениям адресное пространство размером 2 ГБ, также известное как виртуальное адресное пространство пользователя. Все потоки одного приложения содержатся в одном виртуальном адресном пространстве пользователя. Оставшиеся 2 ГБ пространства резервируются операционной системой (эту область памяти также называют адресным пространством ядра). Windows 2000 Server и более поздние выпуски операционных систем, включая Windows Server 2003, содержат параметр boot.ini, позволяющий приложениям получать

доступ к 3 ГБ адресного пространства и ограничивающий адресное пространство ядра размером в 1 ГБ.



В следующей версии Microsoft SQL Server эта возможность будет удалена. Не используйте ее при работе над новыми приложениями и как можно быстрее измените приложения, в которых она в настоящее время используется.

Расширения AWE позволяют 32-разрядным приложениям получать доступ ко всему объему физической памяти, которым располагает операционная система. Такой результат достигается путем сопоставления подмножества объемом до 64 ГБ с адресным пространством пользователя. Сопоставление буферного пула приложения с памятью, сопоставленной AWE, осуществляется при помощи таблиц виртуальной памяти Windows.

Чтобы обеспечить поддержку адресного пространства пользователя объемом 3 ГБ, необходимо добавить в файл boot.ini параметр **/3gb**, после чего следует перезагрузить компьютер, чтобы параметр **/3gb** вступил в силу. Установка этого параметра позволяет потокам пользовательского приложения получать доступ к 3 ГБ адресного пространства и резервирует 1 ГБ для операционной системы.

#### Примечание

Если компьютер имеет свыше 16 ГБ физической памяти, то операционная система требует 2 ГБ адресного пространства процесса и, следовательно, может поддерживать

только 2 ГБ адресного пространства пользователя. Чтобы расширения AWE использовали диапазон памяти свыше 16 ГБ, удалите параметр **/3gb** из файла boot.ini. Если его не удалить, то система не сможет обращаться к памяти свыше 16 ГБ.

## Динамическое управление памятью

Поведение управления памятью в компоненте Microsoft SQL Server Database Engine по умолчанию заключается в использовании необходимого объема памяти в том количестве, чтобы исключить нехватку памяти в системе. Компонент Database Engine осуществляет это при помощи API уведомления памяти в Microsoft Windows.

Виртуальное адресное пространство SQL Server может быть разделено на две отдельных области: пространство, занятое буферным пулом, и остальное. Если включен механизм AWE, буферный пул может располагаться в сопоставленной памяти AWE, предоставляя дополнительное пространство для страниц базы данных.

Буферный пул служит основным источником размещения памяти SQL Server. Внешние компоненты, которые расположены внутри процесса SQL Server, такие как объекты COM и средства, не знающие о функциях управления памятью SQL Server, используют память вне виртуального адресного пространства, занятого буферным пулом.

При запуске SQL Server вычисляется размер виртуального адресного пространства для буферного пула на основании числа параметров, таких как объем физической памяти в системе, число потоков сервера и различные параметры запуска. SQL Server резервирует вычисляемый объем виртуального адресного пространства процесса для буферного пула, но занимает (фиксирует) только необходимый объем физической памяти для текущей нагрузки.

После этого экземпляр продолжает занимать память по мере необходимости для поддержания рабочей нагрузки. По мере того как больше пользователей подключается и выполняет запросы, SQL Server занимает дополнительную физическую память. Экземпляр SQL Server продолжает занимать физическую память до тех пор, пока не достигнет значения, указанного параметром **max server memory**, или до тех пор, пока Windows не сообщит о том, что свободной памяти не осталось; он освобождает память, если она занята больше, чем указано в параметре **min server memory**, а Windows сообщает о том, что свободной памяти не хватает.

По мере того как на компьютере, где запущен экземпляр SQL Server, запускаются другие приложения, они используют память, и объем свободной физической памяти падает ниже цели SQL Server. Экземпляр SQL Server регулирует использование памяти. Если другое приложение остановлено и память становится доступной, экземпляр SQL Server увеличивает размер своего размещения памяти. SQL Server может освободить и занять несколько мегабайтов памяти в секунду, что позволяет быстро приспособиться к изменениям размещения памяти.

## Параметры настройки min server memory и max server memory

Параметры конфигурации **min server memory** и **max server memory** устанавливают



верхний и нижний пределы объема памяти, занятого буферными пулами компонента SQL Server Database Engine. Буферный пул не выделяет объем памяти, указанный в параметре **min server memory**, немедленно. Он начинает расти от объема, необходимого для инициализации. По мере увеличения рабочей нагрузки на компонент Database Engine выделение памяти продолжается. Буферный пул не освобождает занятую память до тех пор, пока не достигнет размера, указанного в параметре **min server memory**. Как только достигнут объем, указываемый параметром **min server memory**, буферный пул применяет стандартный алгоритм выделения и освобождения памяти по мере необходимости. Единственное отличие заключается в том, что буферный пул никогда не освобождает память ниже предела, указываемого параметром **min server memory**, и никогда не занимает объем больше предела, указанного в параметре **max server memory**.

#### Примечание

Сам SQL Server как процесс занимает больше памяти, чем указано в параметре **max server memory**. И внутренние, и внешние компоненты могут занимать память за пределами буферного пула, что также входит в ее общий расход, однако буферный пул обычно составляет наибольшую часть общего объема памяти, потребляемого SQL Server.

Объем памяти, занимаемой компонентом Database Engine, напрямую зависит от рабочей нагрузки на экземпляр. Экземпляр SQL Server, обрабатывающий не слишком много запросов, может и не достигнуть предела, указываемого параметром **min server memory**.

Если и для параметра **min server memory**, и для параметра **max server memory** указано одно и то же значение, то, как только выделенная память достигает этого значения, компонент Database Engine прекращает динамическое выделение и освобождение памяти для буферного пула.

Если экземпляр SQL Server работает на компьютере, на котором часто запускаются и останавливаются другие приложения, выделение и освобождение памяти экземпляром SQL может замедлить запуск других приложений. Кроме того, если SQL Server является одним из нескольких серверных приложений, выполняющихся на компьютере, системным администраторам может понадобиться возможность контроля памяти, выделяемой для SQL Server. В этом случае параметры **min server memory** и **max server memory** помогут управлять объемами памяти, потребляемой SQL Server.

Дополнительные сведения см. в разделе [Параметры памяти сервера](#).

Параметры **min server memory** и **max server memory** указываются в мегабайтах.

## Требования к объему памяти для хранения объектов SQL Server

Следующая таблица содержит примерные сведения об объеме памяти, занимаемой различными объектами SQL Server. Эти значения являются оценочными и могут изменяться в зависимости от среды и способов создания объектов. Рядом элементов SQL Server 2005 управляет при помощи методов, заметно отличающихся от методов, используемых в предыдущих версиях серверов баз данных.

	Схема SQL Server 2000	SQL Server 2005	SQL Server 2008
Блокировка	64 байта + 32 байта на владельца	64 байта + 32 байта на владельца	64 байта + 32 байта на владельца
Открытая база данных	3 924 байта + 1 640 байт на файл и 336 байт на файловую группу	Неприменимо для SQL Server 2005	Неприменимо для SQL Server 2008
Открытый объект	256 байт + 1724 байта на открытый индекс объекта	Неприменимо для SQL Server 2005	Неприменимо для SQL Server 2008
Соединение пользователя	12 КБ + (3 * размер_сетевого_пакета)	Примерно 3 *размер_сетевого_пакета + 94 КБ	Примерно 3 *размер_сетевого_пакета + 94 КБ

Размер сетевого пакета — это размер пакетов схемы табличных данных (TDS), которые используются для обмена данными между приложениями и SQL Server Database Engine.

По умолчанию размер пакета равен 4 КБ, а его настройка осуществляется с помощью параметра конфигурации network packet size.

Если разрешено использование режима MARS, то пользовательское соединение занимает примерно  $(3 + 3 * \text{число\_логических\_соединений}) * \text{размер\_сетевого\_пакета} + 94 \text{ КБ}$ .

## Управление буферами

Главное назначение базы данных SQL Server — хранение и поиск данных, поэтому интенсивное использование операций дискового ввода-вывода — это основное свойство компонента Database Engine. А так как дисковые операции ввода-вывода могут потреблять много ресурсов и требовать относительно длительного времени для выполнения, SQL Server обращает огромное внимание на рациональное использование операций ввода-вывода. Управление буфером — это ключевой компонент в достижении этой рациональности. Компонент управления буфером состоит из двух механизмов: диспетчер буферов для доступа и обновления страниц базы данных и буферный кэш (известный как буферный пул) для сокращения операций ввода-вывода файла базы данных.

## Принцип работы управления буфером

Буфер — это 8-килобайтовая (КБ) страница в памяти такого же размера, что и страница данных или индекса. Буферный кэш делится на 8-килобайтовые страницы. Диспетчер буферов содержит функции чтения страниц данных или индекса из файлов базы данных на диске в буферный кэш и записывает измененные страницы обратно на диск. Страница остается в буферном кэше до тех пор, пока диспетчеру буферов не понадобится область буфера для считывания дополнительных данных. Данные

записываются обратно на диск, только если они были изменены. Данные в буферном кэше могут измениться несколько раз, прежде чем будут сохранены обратно на диске. Дополнительные сведения см. в разделах [Считывание страниц](#) и [Запись страниц](#).

Во время запуска SQL Server вычисляет размер виртуального адресного пространства для буферного кэша, основанного на ряде параметров, например: количество физической памяти в системе, указанное максимальное количество потоков сервера и различные параметры запуска. SQL Server резервирует этот объем виртуального адресного пространства процесса (целевой объем памяти) в буферном кэше, но получает (фиксирует) только необходимый объем физической памяти, требуемый для текущей загрузки. Можно запросить столбцы **bpool\_commit\_target** и **bpool\_committed** в представлении каталога [sys.dm\\_os\\_sys\\_info](#), чтобы получить количество зарезервированных страниц в качестве указателя памяти и количество зафиксированных страниц в буферном кэше соответственно.

Интервал между загрузкой SQL Server и получением буфером кэша указателя памяти называется линейным нарастанием. В течение этого времени читаемые запросы заполняют буфер по мере заполнения. Например, запрос чтения одной страницы заполняет одну страницу буфера. Это означает, что линейное нарастание зависит от числа и типа запросов клиента. Линейное нарастание ускорено благодаря преобразованию запросов чтения одной страницы в запросы, одновременно работающие с восемью страницами. Это позволяет линейному нарастанию завершить операцию намного быстрее, особенно на машинах с большим объемом памяти.

Поскольку диспетчер буферов использует большой объем памяти в процессе SQL Server, он взаимодействует с диспетчером памяти, чтобы дать возможность другим компонентам использовать его буфер. Диспетчер буферов взаимодействует прежде всего со следующими компонентами.

- Диспетчер ресурсов управляет полным использованием памяти и использованием адресного пространства на 32-разрядных платформах.
- Диспетчер базы данных и операционная система SQL Server (SQLOS) для низкоуровневых операций файлового ввода-вывода.
- Диспетчер журнала для упреждающего ведения журнала.

## Поддерживаемые возможности

Диспетчер буферов поддерживает следующие возможности:

- Он учитывает неоднородный доступ к памяти (NUMA). Страницы буферного кэша распределены между узлами оборудования NUMA, которые позволяют потоку обращаться к странице буфера, расположенной на локальном узле NUMA, а не во внешней памяти. Дополнительные сведения см. в разделе [Как SQL Server поддерживает архитектуру NUMA](#). Чтобы понять, как распределяются страницы памяти буферного кэша в конфигурации NUMA, см. раздел [Расширение и сжатие буферного пула в конфигурации с неоднородным доступом к памяти \(NUMA\)](#).
- Диспетчер буферов поддерживает технологию памяти с «горячей» заменой, которая позволяет пользователям добавлять физическую память, не перезапуская сервер. Дополнительные сведения см. в разделе [Память с «горячей» заменой](#).

- Диспетчер буферов поддерживает динамическое распределение памяти на 32-разрядных платформах Microsoft Windows XP и Windows 2003, если включены расширения AWE. Динамическое распределение памяти позволяет компоненту Database Engine эффективно получать и освобождать память в буферном кэше, чтобы поддерживать текущую рабочую нагрузку. Дополнительные сведения см. в разделе [Динамическое управление памятью](#).
- Диспетчер буферов поддерживает большие страницы на 64-разрядных платформах. Размер страницы зависит от версии Windows. Дополнительные сведения см. в документации по Windows.
- Диспетчер буферов обеспечивает дополнительную диагностику, выполняемую с помощью динамических представлений управления. Можно использовать эти представления, чтобы контролировать различные ресурсы операционной системы, характерные для SQL Server. Например, можно использовать представление [sys.dm\\_os\\_buffer\\_descriptors](#), чтобы контролировать страницы в буферном кэше. Дополнительные сведения см. в разделе [Динамические административные представления и функции, связанные с операционной системой SQL Server \(Transact-SQL\)](#).

## Операции дискового ввода-вывода

Диспетчер буферов выполняет только чтение и запись в базу данных. Другие операции над файлами и базами данных, такие как открытие, закрытие, расширение и сжатие, выполняются диспетчером базы данных и компонентами диспетчера файлов.

Дисковые операции ввода-вывода, выполняемые диспетчером буфера, имеют следующие характеристики.

- Все операции ввода-вывода выполняются асинхронно, что позволяет вызывающему потоку продолжать обработку во время выполнения операции ввода-вывода в фоновом режиме.
- Все операции ввода-вывода происходят в вызывающих потоках, если не используется параметр [affinity I/O](#). Параметр «affinity I/O mask» привязывает операцию дискового ввода-вывода SQL Server к определенному подмножеству ЦП. В средах высокоскоростной обработки транзакций (OLTP) SQL Server данное расширение может улучшать производительность потоков SQL Server, выдающих вводы-выводы.
- Операции ввода-вывода нескольких страниц выполняются с операциями ввода-вывода с разбросом, что позволяет передавать данные из прерывающихся областей памяти. Это означает, что SQL Server может быстро заполнить или записать на диск буферный кэш, предотвращая множество физических запросов операций ввода-вывода.

### *Длительные запросы операций ввода-вывода*

Диспетчер буферов сообщает о любых запросах операций ввода-вывода, которые не были выполнены в течение 15 секунд. Это помогает системному администратору различать ошибки SQL Server и ошибки подсистемы ввода-вывода. Появляется сообщение [Ошибка 833](#), и в журнал ошибок SQL Server записывается следующее:

SQL Server обнаружил %d запросов ввода-вывода, выполняющихся более %d секунд, в файле [%ls] базы данных [%ls] (%d). Дескриптор файла ОС — 0x%p. Смещение последней длительной операции ввода-вывода — %#016I64x.

Длительная операция ввода-вывода может быть чтением или записью; в настоящий

момент это не указывается в сообщении. Сообщение о длительной операции ввода-вывода является предупреждением, а не ошибкой. Они не указывают на неполадки SQL Server. Сообщения помогают системному администратору быстрее находить причину длительного времени отклика SQL Server и распознавать ошибки, происходящие вне средств SQL Server. Также они не требуют никакого действия, но системный администратор должен выяснить, почему выполнение запроса ввода-вывода заняло столько времени и оправдано ли это.

### ***Причины длительных запросов операций ввода-вывода***

Сообщение о длительной операции ввода-вывода может указывать, что ввод-вывод постоянно блокируется и никогда не будет завершен (потерянные операции ввода-вывода) или еще не завершен. Из сообщения невозможно узнать, какой сценарий имеет место, хотя потерянные операции ввода-вывода будут часто приводить ко времени ожидания блокировки.

Длительные операции ввода-вывода часто имеют рабочую нагрузку SQL Server, которая является слишком интенсивной для дисковой подсистемы. Неуправляемая дисковая подсистема может быть получена в следующих случаях.

- В журнале ошибок появляются несколько сообщений о длительных операциях ввода-вывода во время большой рабочей нагрузки SQL Server.
- Счетчики системного монитора показывают большую задержку, длинные очереди или отсутствие времени простоя диска.

Длительные операции ввода-вывода также могут быть вызваны компонентом в пути ввода-вывода (например драйвером, контроллером или программой ПЗУ), постоянно откладывающим обслуживание старого запроса ввода-вывода в пользу обслуживания более новых запросов, которые ближе к текущей позиции головки диска. Основная методика обработки запросов, которые наиболее близки к текущей позиции головки для чтения-записи, называется «элеваторный поиск». Это довольно сложно отследить с помощью системного монитора Windows (PERFMON.EXE), так как большинство операций ввода-вывода выполняется быстро. Длительные операции ввода-вывода могут усугубляться рабочими нагрузками, которые выполняют большой объем операций последовательного ввода-вывода, например: создание резервных копий и восстановление, просмотр таблиц, сортировку, создание индексов, массовую загрузку и очистку файлов.

Изолированные длительные операции ввода-вывода, которые не связаны с любой из указанных выше причин, могут быть вызваны ошибками драйвера или оборудования. Журнал системных событий может содержать связанные события, которые помогают выявить ошибку.

### ***Определение ошибки***

Страницы базы данных могут использовать один из двух дополнительных механизмов, чтобы обеспечить целостность страницы от момента записи на диск до чтения: защита от разрыва страницы и защита контрольной суммой. Эти механизмы позволяют отдельному методу проверить правильность не только хранения данных, но и компонентов оборудования, например контроллеров, драйверов, кабелей и даже

операционной системы. Защита присоединяется к странице как раз перед ее записью на диск и выполняет проверку после чтения страницы с диска.

### *Защита от разрыва страницы*

Защита от разрыва страницы, включенная в SQL Server 2000, является прежде всего способом обнаружения поврежденных страниц из-за сбоев питания. Например, неожиданный сбой питания может оставить только часть страницы, записанной на диск. Когда используется защита от разрыва страницы, в конце каждого 512-байтового сектора на странице помещается 2-битовая подпись (исходные два бита копируются в колонтитул). Подпись чередуется двоичными числами 01 и 10 после каждой записи, так что всегда можно определить, когда на диск записана только часть секторов. Если бит имеет неверное состояние, когда читается страница, значит она была записана неправильно и обнаружен разрыв страницы. Защита от разрыва страницы использует минимальные ресурсы, однако она не обнаруживает все ошибки, вызванные аппаратными сбоями диска.

### *Защита контрольной суммой*

Защита контрольной суммой, включенная в SQL Server 2005, обеспечивает более надежную проверку целостности данных. Контрольная сумма рассчитывается для данных каждой записанной страницы и сохраняется в колонтитуле. Всякий раз, когда страница с сохраненной контрольной суммой читается с диска, компонент Database Engine повторно вычисляет контрольную сумму для данных страницы и вызывает ошибку 824, если новая контрольная сумма отличается от сохраненной. Защита контрольной суммой может перехватить больше ошибок, чем защита от разрыва страницы, потому что она учитывает каждый байт страницы, однако она более ресурсоемкая. Когда защита контрольной суммой активирована, ошибки, вызванные сбоями питания и поврежденным оборудованием или программой ПЗУ, могут быть обнаружены во время чтения страницы с диска диспетчером буферов.

Вид используемой защиты страницы является атрибутом базы данных, содержащей страницу. Защита контрольной суммой задана по умолчанию для баз данных, созданных в SQL Server 2005 и более поздних версиях. Механизм защиты страницы указывается во время создания базы данных и может быть изменен с помощью инструкции ALTER DATABASE. Установку защиты текущей страницы можно определить с помощью запроса значения столбца **page\_verify\_option** в представлении каталога [sys.databases](#) или свойства **IsTornPageDetectionEnabled** функции [DATABASEPROPERTYEX](#). Если установка защиты страницы изменена, новая установка сразу не влияет на всю базу данных. Вместо этого страницы принимают текущий уровень защиты базы данных во время следующей записи. Это означает, что база данных может состоять из страниц с различными видами защиты.

### *Считывание страниц*

Процесс ввода-вывода экземпляра компонента SQL Server Database Engine включает физические и логические операции чтения. Логическое чтение выполняется каждый раз, когда компонент Database Engine запрашивает страницу из [буферного кэша](#). Если в

этот момент страница не находится в кэше, то сначала при помощи операции физического чтения она копируется в него с диска.

Запросы к экземпляру компонента Database Engine на чтение страниц управляются реляционным механизмом, а их дополнительную оптимизацию выполняет подсистема хранилища. Реляционный механизм подбирает наиболее эффективный метод доступа (просмотр таблицы, просмотр индекса или чтение по ключу). Компонент метода доступа и диспетчер буферов подсистемы хранилища определяют общий шаблон считывания и оптимизируют операции чтения для реализации метода доступа. Поток, выполняющий пакет, планирует операции считывания.

### **Упреждающее чтение**

Компонент Database Engine поддерживает механизм оптимизации производительности, называемый упреждающим чтением. Он заключается в том, что система пытается предугадать, какие именно страницы данных и индексов понадобятся для плана выполнения запроса, и помещает эти страницы в буферный кэш, прежде, чем в них возникнет реальная необходимость. Это позволяет распараллелить операции вычисления и ввода-вывода, полностью задействовав процессорные и дисковые мощности.

Механизм упреждающего чтения позволяет компоненту Database Engine считывать из одного файла до 64 последовательных страниц (512 КБ). Эта операция выполняется как единая последовательность разборки-сборки для соответствующего числа буферов в буферном кэше (возможно, расположенных не последовательно). Если какая-либо из страниц этого диапазона уже присутствует в буферном кэше, то соответствующая страница по окончании считывания будет удалена из памяти. Кроме того, диапазон страниц может быть «усечен» с начала или с конца, если соответствующие страницы уже находятся в кэше.

Упреждающее чтение бывает двух видов: для страниц данных и для страниц индексов.

### **Чтение страниц данных**

Просмотр таблиц для чтения страниц данных выполняется в компоненте Database Engine весьма эффективно. Страницы карты распределения индекса (IAM) в базе данных SQL Server содержат указатели на экстенды, занятые таблицей или индексом.

Подсистема хранилища может считывать IAM для построения отсортированного списка адресов на диске, которые необходимо считать. Это позволяет подсистеме хранилища оптимизировать операции ввода-вывода выполняемые последовательно в виде больших операций чтения на основе их размещения на диске. Дополнительные сведения о страницах IAM см. в разделе [Управление дисковым пространством, занятым объектами](#).

### **Считывание страниц индекса**

Подсистема хранилища считывает страницы индекса последовательно, в порядке значений ключей. На приведенном ниже рисунке показан пример упрощенного представления набора конечных страниц, которое содержит набор ключей и промежуточный узел индекса, сопоставляющий концевые страницы. Дополнительные сведения о структуре страниц в индексе см. в разделе [Структуры кластеризованного индекса](#).



**Промежуточный узел индекса:**

AAA: 504
AME: 505
AZE: 527
BIK: 528
CAF: 544
DEE: 556
DZS: 575
EMA: 576

**Конечные узлы:**

Стр. 504 AAA AFA AJE	Стр. 505 AME AOP ARN	Стр. 527 AZE BAB BGA	Стр. 528 BIK CAA CAE	Стр. 544 CAF DAC DDO	Стр. 556 DEE DMA DRT	Стр. 557 DZS EAU EGE	Стр. 576 EMA EMA ERU
-------------------------------	-------------------------------	-------------------------------	-------------------------------	-------------------------------	-------------------------------	-------------------------------	-------------------------------

Подсистема хранилища использует данные промежуточной страницы индекса выше конечного уровня для организации последовательного упреждающего чтения страниц, содержащих значения ключей. При обработке запроса на все ключи от «ABC» до «DEF» вначале подсистема хранилища считывает страницу индекса выше конечной страницы. Однако он не просто последовательно считывает все страницы данных от 504-й до 556-й, то есть последней, на которой находятся значения ключей заданного диапазона. Вместо этого подсистема хранилища просматривает промежуточную страницу индекса и строит список конечных страниц для считывания. Затем подсистема хранилища планирует все операции ввода-вывода в порядке следования ключей. Кроме того, подсистема хранилища определяет, что страницы 504/505 и 527/528 являются смежными, и считывает содержащиеся на них данные за одну операцию, после чего выполняет сборку данных. Если в последовательной операции необходимо получить много страниц, то планируется целый блок операций чтения за один раз. По завершении этого подмножества операций чтения планируется такое же количество новых, и так до тех пор, пока все нужные страницы не будут запрошены.

Затем подсистема хранилища использует упреждающую выборку для ускорения поиска в таблицах по некластеризованным индексам. Конечные строки в некластеризованном индексе содержат указатели на строки данных, в которых хранятся все определенные значения ключей. По мере чтения по конечным страницам некластеризованного индекса подсистема хранилища также начинает планирование асинхронных операций чтения строк данных по уже полученным указателям на них. Это позволяет приступить к извлечению строк данных из базовой таблицы еще до завершения просмотра некластеризованного индекса. Этот процесс выполняется независимо от наличия в таблице кластеризованного индекса. SQL Server Enterprise использует упреждающую выборку в большей степени, чем другие выпуски SQL Server, выполняя упреждающее чтение большего числа страниц. Уровень упреждающей выборки нельзя настраивать ни в одном из выпусков. Дополнительные сведения о некластеризованных индексах см. в разделе [Структуры некластеризованных индексов](#).

**Расширенный просмотр**

Функция расширенного просмотра в выпуске SQL Server Enterprise позволяет нескольким задачам совместно выполнять полный просмотр таблиц. Если план



выполнения инструкции Transact-SQL требует просмотра страниц данных таблицы, а компонент Database Engine обнаружил, что таблица уже просматривается другим планом выполнения, то вторая операция просмотра будет присоединена к первой в том месте, где она в данный момент находится. Компонент Database Engine считывает по странице за раз и передает полученные строки каждой считываемой страницы обоим планам выполнения. Это продолжается до тех пор, пока не будет достигнут конец таблицы.

На этой стадии первый план выполнения получит полные результаты просмотра, а второй план до сих пор не получил страницы данных, находящиеся ранее места присоединения. Поэтому для него производится возврат к первой странице данных, и просмотр повторяется до этой точки. Таким образом, можно объединять любое количество просмотров. Компонент Database Engine продолжает циклическое считывание страниц данных до тех пор, пока не будут завершены все операции просмотра. Этот механизм называется также «кольцевым просмотром» и демонстрирует, почему порядок строк, возвращаемых в результате выполнения инструкции SELECT, без предложения ORDER BY не гарантирован.

Например, допустим, что некая таблица содержит 500 000 страниц. Пользователь UserA выполняет инструкцию Transact-SQL, которая требует просмотра таблицы. По первому запросу просмотрено уже 100 000 страниц, после чего пользователь UserB выполняет еще одну инструкцию Transact-SQL, которая требует просмотра той же таблицы. Для считывания после 100 001-ой страницы компонент Database Engine запланирует всего одну серию запросов и будет возвращать полученные строки обоим планам выполнения. По достижении 200 000-ой страницы пользователь UserC выполняет еще одну инструкцию Transact-SQL, которая требует просмотра той же таблицы. Начиная с 200 001-ой страницы, компонент Database Engine будет передавать строки каждой считываемой страницы всем трем просмотрам. После считывания 500 000-ой строки просмотр для пользователя UserA завершается, а для пользователей UserB и UserC чтение снова начинается с первой страницы. Когда компонент Database Engine доходит до 100 000-ой страницы, просмотр для пользователя UserB завершается. Просмотр для пользователя UserC продолжает выполняться до тех пор, пока не будет достигнута 200 000-ая страница. Только после этого операция просмотра для всех пользователей будет завершена.

Без возможности расширенного просмотра все пользователи были бы вынуждены конкурировать за буферное пространство, что вызвало бы значительную нагрузку на дисковый накопитель. Затем те же страницы считывались бы каждым из пользователей, вместо того чтобы загрузить их один раз для нескольких пользователей.

### ***Запись страниц***

Операции ввода-вывода, выполняемые экземпляром компонента Database Engine, включают в себя операции логической и физической записи. Логическая запись выполняется при изменении данных страницы в [буферном кэше](#). Физическая запись производится при записи страницы из буферного кэша на диск.

Если какая-либо из страницы в буферном кэше изменилась, она не записывается сразу

на диск, а помечается как грязная. Это означает, что до момента ее физической записи на диск к странице применялась одна или несколько операций логической записи. При каждой логической операции записи в кэш журнала, который записывает изменения, добавляется запись журнала транзакций. Записи журнала должны быть записаны на диск до того, как связанная с ними «грязная» страница будет удалена из буферного кэша и записана на диск. SQL Server использует метод, называемый упреждающим ведением журнала, который предотвращает запись «грязных» страниц до записи на диск связанных с ними логических записей. Это особенно важно для правильной работы диспетчера восстановления. Дополнительные сведения см. в разделе [Журнал транзакций с упреждающей записью](#).

На следующем рисунке показан процесс записи измененной страницы данных.



При записи страницы диспетчер буферов производит поиск «грязных» страниц, которые расположены рядом с ней и могут быть включены в ту же операцию записи. Соседние страницы обладают последовательными идентификаторами и извлекаются из одного и того же файла. Эти страницы необязательно должны располагаться в памяти последовательно. Поиск в обоих направлениях продолжается до тех пор, пока не произойдет одно из следующих событий:

- Найдена чистая страница.
- Найдено 32 страницы.
- Найдена «грязная» страница, номер LSN которой еще не записан в журнал.
- Найдена страница, которая не может быть немедленно заблокирована.

Таким образом, весь набор страниц может быть записан на диск всего одной операцией записи.

Непосредственно перед записью страницы к ней применяется та форма защиты страниц, которая указана в базе данных. При защите страницы от разрыва она должна быть кратковременно заблокирована для операций ввода-вывода в режиме EX (монопольный режим), поскольку защита от разрыва изменяет страницы, делая ее недоступной для чтения остальными потоками. Если применяется защита страницы по контрольной сумме или база данных не использует защиту страниц, она блокируется для операций ввода-вывода в режиме UP (блокировка на обновление). Эта блокировка предотвращает изменение страницы во время записи, но разрешает к ней доступ для чтения. Дополнительные сведения о параметрах защиты операций дискового ввода-вывода см. в разделе [Управление буферами](#).

«Грязная» страница записывается на диск одним из трех способов.

- Отложенная запись

Модуль отложенной записи — это системный процесс, который освобождает буферы, удаляя из буферного кэша редко используемые страницы. «Грязные» страницы записываются на диск первыми.

- **Активная запись**

Во время процесса активной записи происходит запись страниц данных, связанных с операциями, не регистрируемыми в журнале (например массовой вставкой или SELECT INTO). Это позволяет создавать и записывать новые страницы параллельно. Поэтому вызывающая операция не ожидает окончания всей операции, прежде чем записать страницу на диск.

- **Контрольная точка**

Заключается в том, что процесс контрольных точек производит периодический просмотр буферного кэша на наличие буферов со страницами определенной базы данных и запись всех «грязных» страниц на диск. Этот метод позволяет экономить время во время последующего восстановления, создавая точку, в которой все «грязные» страницы гарантированно записаны на диск. Пользователь может запросить создание контрольной точки, выполнив команду CHECKPOINT. Кроме того, компонент Database Engine может автоматически создавать контрольные точки, ориентируясь по размеру журналов и времени, прошедшему с момента создания последней контрольной точки. Помимо этого, контрольная точка создается при возникновении событий определенного рода, например когда файл журнала или файл данных добавляется или удаляется из базы данных либо при остановке экземпляра SQL Server. Дополнительные сведения см. в разделе [Контрольные точки и активная часть журнала](#).

В режимах отложенной записи, активной записи и контрольной точки ожидание завершения операций ввода-вывода не производится. Они всегда выполняются асинхронно и параллельно с другими операциями, откладывая проверку успешности завершения операций ввода-вывода на более поздний срок. Это позволяет SQL Server максимально эффективно использовать ресурсы процессора и устройств ввода-вывода для соответствующих задач.

## Управление памятью для больших баз данных

SQL Server для поддержки физической памяти очень большого объема использует функции API расширений AWE. В SQL Server поддерживается до 64 ГБ физической памяти для 32-разрядных версий операционных систем Microsoft Windows:

Microsoft Windows XP Professional; Windows 2000 Standard Edition; Windows 2000 Advanced Server; Windows 2000 Datacenter Server; Windows Server 2003 Enterprise Edition и Windows Server 2003 Datacenter Edition.

SQL Server динамически распределяет расширенную память AWE при работе в любом из выпусков операционной системы Windows Server 2003. Иначе говоря, буферный пул может динамически управлять памятью AWE для выравнивания использования памяти SQL Server в соответствии с общими системными требованиями.

Расширения AWE позволяют преодолеть ограничения, присущие 32-разрядным приложениям, которые не могут производить доступ к адресному пространству процесса объемом более 4 ГБ. (32-разрядный указатель не может содержать адрес памяти больше 4 ГБ).

При помощи расширений AWE приложения могут непосредственно, без подкачки, резервировать память вплоть до максимального объема физической памяти, допустимого операционной системой. Применение расширений AWE позволяет SQL

Server кэшировать больше данных, не считывая их из системных файлов подкачки на диске. Это обеспечивает улучшение производительности благодаря более быстрому доступу к данным и снижает частоту обращений к диску.

#### Примечание

Для 64-разрядных приложений расширения AWE не нужны, так как в них доступ к памяти не ограничивается объемом 4 ГБ.

Дополнительные сведения об API расширений AWE см. на [веб-узле MSDN](#). Перейдите по ссылке и выполните поиск по словосочетанию «Address Windowing Extensions».

Дополнительные сведения о работе SQL Server при неоднородном доступе к памяти (NUMA) см. в разделе [Основные сведения о неоднородном доступе к памяти](#).

## Использование расширений AWE

В следующей версии Microsoft SQL Server эта возможность будет удалена. Не используйте ее при работе над новыми приложениями и как можно быстрее измените приложения, в которых она в настоящее время используется.

Службы SQL Server используют Microsoft API расширений AWE для поддержки очень большого объема физической памяти. Службы SQL Server могут адресовать до 64 гигабайт (ГБ) памяти в операционных системах Microsoft Windows 2000 Server и Microsoft Windows Server 2003.

#### Примечание

Поддержка AWE доступна только в выпусках SQL Server Enterprise, Standard и Developer для 32-разрядных версий SQL Server. Преимущества расширенной памяти AWE недоступны для служб Службы Analysis Services. Если объем доступной физической памяти меньше, чем пространство виртуальных адресов пользовательского режима, то AWE включить нельзя.

Стандартная 32-разрядная адресация может отобразить максимум 4 ГБ памяти. Поэтому стандартные адресные пространства 32-разрядных процессов ограничены 4 ГБ. По умолчанию на 32-разрядных версиях операционных систем Microsoft Windows 2 ГБ зарезервированы для операционной системы, и 2 ГБ доступны для приложений. Если в файле Boot.ini указан параметр **/3GB** для выпусков Windows 2000 Advanced Server, операционная система зарезервирует только 1 ГБ адресного пространства, и приложения смогут обращаться к 3 ГБ физической памяти. Дополнительные сведения о параметре **/3gb** см. в разделе [Адресное пространство процесса](#).

Расширение AWE является набором расширений к функциям управления памятью Windows, которые позволяют приложениям обращаться к большему количеству памяти, чем 2-3 ГБ, которая доступна через стандартную 32-разрядную адресацию. Расширение AWE позволяет приложениям запрашивать физическую память и затем динамически отображать представление не разделенной на страницы памяти на 32-разрядное адресное пространство. Хотя 32-разрядное адресное пространство ограничено 4 ГБ, не разделенная на страницы память может быть намного больше. Это позволяет приложениям, интенсивно использующим память, таким как большие системы баз

данных, обращаться к большему объему памяти, чем поддерживается в 32-разрядном адресном пространстве.

Перед настройкой операционной системы для использования расширений AWE продумайте следующее:

- Расширения AWE позволяют распределять физическую память за пределы 4 ГБ на 32-разрядной архитектуре. Расширения AWE должны использоваться, только если объем доступной физической памяти превышает размеры виртуального адресного пространства пользовательского режима.
- Для поддержки более 4 ГБ физической памяти в 32-разрядных операционных системах необходимо добавить параметр **/paе** в файл Boot.ini и перезагрузить компьютер. Дополнительные сведения см. в документации Windows.

#### Примечание

В Windows Server 2003 PAE автоматически включено, только если сервер использует устройства памяти с «горячей» заменой. В этом случае не нужно использовать переключатель **/PAE** в системе, которая настроена на использование памяти с «горячей» заменой. Во всех других случаях для поддержки более 4 ГБ физической памяти необходимо добавить параметр **/PAE** в файл boot.ini.

- Если на компьютере доступно больше, чем 16 ГБ физической памяти, операционной системе в виртуальном адресном пространстве необходимо 2 ГБ для системных целей, и поэтому может поддерживаться только 2 ГБ виртуального адресного пространства пользовательского режима. Чтобы операционная система использовала диапазон памяти свыше 16 ГБ, удалите параметр **/3gb** из файла boot.ini. Если параметр указан, операционная система не сможет использовать физическую память выше 16 ГБ.

#### Примечание

Буферный пул SQL Server может полностью использовать память AWE, однако только страницы базы данных могут динамически размещаться и выгружаться из виртуального адресного пространства SQL Server и в полной мере использовать память, распределенную через расширения AWE. Расширения AWE не помогают напрямую поддерживать дополнительных пользователей, потоки, базы данных, запросы и другие объекты, которые постоянно находятся в виртуальном адресном пространстве.

Таблицу, перечисляющую значения максимальной памяти сервера, см. в разделе [Архитектура оперативной памяти](#).

## Включение поддержки более 4 ГБ физической памяти

Чтобы включить расширения AWE для Microsoft SQL Server, нужно запустить компонент SQL Server Database Engine под учетной записью Microsoft Windows, которая имеет разрешение Блокировка страниц в памяти, и задать для параметра awe enabled значение 1 с помощью хранимой процедуры sp\_configure.

#### Примечание

Поддержка AWE доступна только в выпусках SQL Server Enterprise, Standard и Developer для 32-разрядных операционных систем. Преимущества расширенной

памяти AWE недоступны для служб Службы Analysis Services. Если объем доступной физической памяти меньше, чем пространство виртуальных адресов пользовательского режима, то AWE включить нельзя.

### ***Закрепление страниц в памяти***

Эта политика определяет, какие учетные записи вправе использовать процесс для хранения данных в физической памяти, что предотвращает страничную запись данных операционной системой в область виртуальной памяти на диск. Параметр Блокировка страниц в памяти в SQL Server по умолчанию имеет значение OFF. Пользователь, обладающий разрешениями администратора, может включить этот параметр вручную с помощью инструмента «Политика группировки Windows» (**gpedit.msc**) и присвоить соответствующее разрешение той учетной записи, под которой запускается SQL Server.

Дополнительные сведения о том, как включить параметр Блокировка страниц в памяти см. в разделе [Как включить параметр «Блокировка страниц в памяти» \(Windows\)](#).

Хотя это и не обязательно, рекомендуется включить закрепление страниц в памяти при использовании 64-разрядных операционных систем. Для 32-разрядных операционных систем разрешение на блокировку страниц в памяти должно быть предоставлено до начала настройки AWE в SQL Server.

### ***Параметр awe enabled***

Чтобы включить AWE для экземпляра SQL Server, задайте с помощью хранимой процедуры `sp_configure` значение 1 для параметра **awe enabled**, затем перезапустите SQL Server. Поскольку AWE включается во время запуска SQL Server и остается включенным вплоть до закрытия SQL Server, SQL Server уведомит пользователей об использовании параметра `awe enabled` путем отправки сообщения «Расширения AWE включены» в журнал ошибок SQL Server. Дополнительные сведения о параметре конфигурации `awe enabled` см. в разделе [Параметр awe enabled](#).

### ***Увеличение пропускной способности в сетевом приложении до предела***

Чтобы оптимизировать использование системной памяти для SQL Server, следует ограничить объем памяти, используемой системой для кэширования файлов. Чтобы ограничить кэш файловой системы, убедитесь, что снят флажок макс. пропускная способность доступа к общим файлам.

Проверка значения параметра в операционной системе.

1. В Панели управления дважды щелкните Сетевые подключения, затем выберите Подключение по локальной сети.
2. На вкладке Общие нажмите кнопку Свойства и выберите Служба доступа к файлам и принтерам сетей Microsoft, затем нажмите кнопку Свойства.
3. Выберите параметр Макс. пропускная способность для сетевых приложений, нажмите кнопку ОК и закройте остальные диалоговые окна.

Дополнительные сведения об использовании AWE см. в документации Windows.



## Активация памяти расширений AWE для SQL Server

В следующей версии Microsoft SQL Server эта возможность будет удалена. Не используйте ее при работе над новыми приложениями и как можно быстрее измените приложения, в которых она в настоящее время используется.

Расширения AWE позволяют 32-разрядным операционным системам обращаться к большим объемам памяти. Расширения AWE предоставляются операционной системой и реализованы с незначительными различиями в операционных системах Microsoft Windows 2000 Server и Windows Server 2003. Расширения AWE включаются с помощью [Параметр awe enabled](#).

### *Использование отображаемой памяти расширений AWE с Windows 2000 Server*

Экземпляры Microsoft SQL Server, выполняющиеся в Windows 2000, распределяют память во время запуска SQL Server согласно следующей последовательности событий.

- Если доступная физическая память оказывается меньше, чем виртуальное адресное пространство пользовательского режима, расширения AWE включить не удастся. В этом случае SQL Server выполняется без использования расширений AWE независимо от значения параметра **awe enabled**.
- Если доступная физическая память больше, чем виртуальное адресное пространство пользовательского режима, то расширения AWE можно включить.
  - Если объем доступной физической памяти превышает значение параметра **max server memory**, экземпляр SQL Server блокирует объем памяти, указанный в параметре **max server memory**.
  - Если доступная физическая память меньше, чем значение параметра **max server memory**, или если параметр **max server memory** не был установлен, экземпляр SQL Server блокирует всю доступную память, кроме 256 МБ.
- После выделения отображаемой памяти расширений AWE она не может быть освобождена до закрытия SQL Server.

### Настройка параметров памяти

Настоятельно рекомендуется устанавливать значение для параметра **max server memory** каждый раз при включении расширений AWE. Когда для параметра **awe enabled** установлено значение 1 (и доступная физическая память больше, чем пространство процесса непривилегированного режима), экземпляры SQL Server, выполняющиеся в Windows 2000, при запуске блокируют почти всю доступную память (или величину **max server memory**, если параметр был задан). Если параметр **max server memory** не установлен, другим приложениям или экземплярам SQL Server будет доступно меньше, чем 128 МБ физической памяти.

Пул расширенной памяти AWE не может быть выгружен в системные файлы подкачки. Если для использования требуется дополнительная физическая память, операционная система Windows должна выгрузить другие приложения, что может воспрепятствовать работе этих приложений.

Во избежание потери производительности других приложений настройте параметр **max server memory** таким образом, чтобы оставалось немного дополнительной свободной

памяти для различных требований других приложений и операционной системы. Можно определить объем памяти, безопасно распределяемый экземплярам компонента SQL Server, установив количество памяти, доступной после того, как были запущены все другие приложения, использующиеся на компьютере.

#### Примечание

В Windows 2000 Server параметр **min server memory** не учитывается для расширений AWE в SQL Server.

Используйте счетчик **Общая память сервера (КБ)** монитора производительности SQL Server, чтобы определить, сколько памяти распределено экземпляром SQL Server, выполняющимся в режиме AWE, или выберите сведения об использовании памяти из **sysperfinfo**.

Дополнительные сведения см. в разделе [Мониторинг использования памяти](#).

### Запуск нескольких экземпляров SQL Server с расширениями AWE

Если сервер работает под управлением Windows 2000, каждый экземпляр должен иметь настройку **max server memory**. Поскольку SQL Server в операционной системе Windows 2000 не поддерживает динамическое выделение отображаемой памяти расширений AWE, рекомендуется установить параметр **max server memory** для каждого экземпляра.

Сумма значений **max server memory** для всех экземпляров должна быть меньше, чем общая физическая память компьютера. Если сумма больше, чем общий объем физической памяти, некоторые из экземпляров не запустятся или будут иметь доступ к меньшему объему памяти, чем указано параметром **max server memory**. Предположим, что компьютер имеет 16 гигабайт (ГБ) физической памяти и три установленных экземпляра SQL Server. Кроме того, параметр **max server memory** установлен на 8 ГБ для каждого экземпляра. Если останутся и перезапустятся все три экземпляра, память будет распределена следующим образом.

1. Первый экземпляр имеет доступ к 8 ГБ физической памяти.
2. Второй экземпляр запускается, но имеет доступ к несколько меньшему объему памяти, чем 8 ГБ (менее 128 МБ).
3. Третий экземпляр запускается в режиме динамической памяти и имеет доступ к 256 МБ или к меньшему объему физической памяти.

Дополнительные сведения см. в разделе [Управление памятью для больших баз данных](#).

### Использование отображаемой памяти расширений AWE с Windows Server 2003

SQL Server поддерживает динамическое выделение памяти расширений AWE в Windows Server 2003. Во время запуска SQL Server резервирует только небольшую часть памяти расширений AWE. По мере появления потребности в дополнительной памяти AWE операционная система динамически выделяет ее SQL Server. Аналогично, если требуется меньшее количество ресурсов, SQL Server может вернуть отображаемую память расширений AWE операционной системе для использования другими процессами или приложениями. Дополнительные сведения о параметре конфигурации **awe enabled** см. в разделе [Параметр awe enabled](#).



Объем поддерживаемой физической памяти вырос с выходом семейства операционных систем Windows Server 2003. Объем физической памяти, доступной расширениям AWE, зависит от используемой операционной системы. В следующем списке приведены сведения о максимальном объеме физической памяти, доступном для каждой операционной системы семейства Windows Server 2003, на момент написания данного раздела.

- Выпуск Windows Server 2003 Standard Edition поддерживает до 4 ГБ физической памяти.
- Выпуск Windows Server 2003 Enterprise Edition поддерживает до 32 ГБ физической памяти.
- Выпуск Windows Server 2003 Datacenter Edition поддерживает до 64 ГБ физической памяти.

### Настройка параметров памяти

SQL Server динамически распределяет память расширений AWE при выполнении в любом из выпусков операционной системы Windows Server 2003. Другими словами, буферный пул может динамически управлять отображаемой памятью расширений AWE (в пределах ограничений параметров **min server memory** и **max server memory**) для балансировки использования памяти SQL Server с общими требованиями к системе.

Когда расширения AWE включены, SQL Server всегда пытается использовать отображаемую память AWE. Это применимо ко всем конфигурациям памяти, включая компьютеры, настроенные на предоставление приложениям менее 3 ГБ адресного пространства в пользовательском режиме.

- Рекомендуется установить AWE как режим памяти по умолчанию для SQL Server, работающего под управлением Windows Server 2003. Для функции памяти с «горячей» заменой необходимо, чтобы расширения AWE были включены во время запуска SQL Server. Дополнительные сведения см. в разделе [Память с «горячей» заменой](#).

#### Примечание

Расширения AWE не являются обязательными и не могут быть настроены в 64-разрядных операционных системах.

- Так как отображаемая память расширений AWE поддерживается в области ниже 3 ГБ, можно определить значения **min server memory** и **max server memory** в пределах диапазона физической памяти или использовать значения по умолчанию для обоих параметров.
- Можно рассмотреть настройку **max server memory** для SQL Server, чтобы гарантировать предоставление дополнительной памяти другим приложениям, работающим на компьютере. Хотя SQL Server может динамически освобождать отображаемую память расширений AWE, текущий объем распределенной отображаемой памяти расширений AWE не может быть переключен на файл подкачки.

Чтобы сделать расширения AWE доступными для экземпляра SQL Server, воспользуйтесь хранимой процедурой **sp\_configure**, чтобы установить параметр **awe enabled** в 1, и затем перезапустите SQL Server.

Дополнительные сведения о параметрах **min server memory** и **max server memory** см. в разделе [Параметры памяти сервера](#).

Прежде чем включить расширения AWE, необходимо настроить политику **Закрепление страниц в памяти**. Дополнительные сведения см. в разделе [Как включить параметр «Блокировка страниц в памяти» \(Windows\)](#).

### Пример

Следующий пример показывает, как активизировать AWE и настроить ограничение 1 ГБ для **min server memory** и 6 ГБ для **max server memory**.

Сначала настройте AWE:

```
sp_configure 'show advanced options', 1
RECONFIGURE
GO
```

```
sp_configure 'awe enabled', 1
RECONFIGURE
GO
```

После перезапуска SQL Server будет записано следующее сообщение в журнал ошибок SQL Server: «Расширения AWE включены.»

Далее настройте память:

```
sp_configure 'min server memory', 1024
RECONFIGURE
GO
```

```
sp_configure 'max server memory', 6144
RECONFIGURE
GO
```

В этом примере настройки памяти устанавливают буферный пул для динамического управления отображаемой памятью расширений AWE между 1 ГБ и 6 ГБ. Если другому приложению потребуется дополнительная память, SQL Server может высвободить выделенную под расширения AWE память в случае ее ненадобности. В этом примере отображаемая память расширений AWE может быть высвобождена только до 1 ГБ.

Динамическая память AWE также позволяет SQL Server увеличивать память, если дополнительная память добавлена к компьютеру, который поддерживает память с «горячей» заменой. Доступная в Windows Server 2003, в выпусках Enterprise Edition и Datacenter Edition, память с «горячей» заменой позволяет добавить память во время работы компьютера. Предположим, что запуск SQL Server под управлением выпуска Windows Server 2003 Enterprise происходит на компьютере с 16 ГБ физической памяти. Операционная система настроена на предоставление приложениям не более 2 ГБ адресного пространства виртуальной памяти; расширения AWE были активированы для SQL Server. Позже системный администратор добавит 16 ГБ памяти во время работы компьютера. Службы SQL Server немедленно распознают дополнительную память и в случае необходимости задействуют ее.

Дополнительные сведения об использовании AWE см. в документации Windows Server 2003.

## Запуск нескольких экземпляров SQL Server с расширениями AWE

Если на одном компьютере работает несколько экземпляров SQL Server и каждый экземпляр использует отображаемую память расширений AWE, необходимо обеспечить правильную работу экземпляров.

Если сервер работает под управлением Windows Server 2003, для каждого экземпляра должен быть определен параметр **min server memory**. Поскольку SQL Server под управлением Windows Server 2003 поддерживает динамическое управление памятью расширений AWE, рекомендуется определить параметр **min server memory** для каждого экземпляра. Так как отображаемая память расширений AWE не может быть переключена на файл подкачки, сумма значений **min server memory** для всех экземпляров должна быть меньше, чем общая физическая память на компьютере.

Параметр **min server memory** не заставляет SQL Server получать минимальный объем памяти при запуске. Память распределяется по требованию, основываясь на рабочей загрузке базы данных. Однако, как только достигнут порог **min server memory**, память не будет освобождаться SQL Server, если это может привести к тому, что SQL Server останется с меньшим количеством памяти, чем указано в параметре. Чтобы гарантировать, что каждому экземпляру выделена память (по крайней мере, соответствующая значению параметра **min server memory**), рекомендуется выполнить загрузку сервера базы данных вскоре после запуска. В ходе нормальной деятельности сервера доступная экземпляру память меняется, но она никогда не станет меньше, чем значение параметра **min server memory** каждого экземпляра.

Можно задать параметр **max server memory** или оставить для него значение по умолчанию. Если оставить для параметра **max server memory** значение по умолчанию, это может привести к конкуренции за память между экземплярами SQL Server.

## Использование кластера отработки отказа с AWE

Если используется кластер отработки отказа SQL Server и память AWE, необходимо гарантировать, что сумма настроек **max server memory** для всех экземпляров будет меньше наименьшего доступного объема физической памяти на любом из серверов в отказоустойчивом кластере. Если у узла отработки отказа физической памяти меньше, чем у исходного, может случиться, что экземпляры SQL Server не запустятся вообще или запустятся с меньшим объемом памяти, чем они имели на исходном узле.

## Память с «горячей» заменой

Версии SQL Server до SQL Server 2005 поддерживали динамическую память, что позволяло SQL Server автоматически регулировать использование памяти при наличии свободной памяти в системе. Однако SQL Server был ограничен объемом памяти, доступной при запуске. Это ограничение было снято в SQL Server 2005.

SQL Server поддерживает технологию памяти с «горячей» заменой в Windows Server 2003, которая позволяет пользователям добавлять физическую память, не перезапуская сервер.

Память с «горячей» заменой требует SQL Server Enterprise и доступна только в 64-

разрядной версии SQL Server, а также в 32-разрядной версии SQL Server с включенным режимом AWE. Память с «горячей» заменой недоступна в 32-разрядной версии SQL Server при выключенном режиме AWE. Память с «горячей» заменой доступна только в выпусках Windows Server 2003 Enterprise Edition и Datacenter Edition. Также необходимо, чтобы поставщик оборудования поддерживал специальное оборудование.

#### Примечание

Чтобы использовать память с «горячей» заменой с 32-разрядной версией SQL Server, следует запустить SQL Server с параметром **-h**. Дополнительные сведения см. в разделе [Использование параметров запуска службы SQL Server](#).

#### Примечание

В Windows Server 2003 PAE автоматически включено, только если сервер использует устройства памяти с «горячей» заменой. В этом случае не нужно использовать переключатель **/PAE** в системе, которая настроена на использование памяти с «горячей» заменой. Во всех других случаях для поддержки более 4 ГБ физической оперативной памяти необходимо добавить параметр **/PAE** в файл boot.ini.

Чтобы физическую память можно было добавить в систему, SQL Server должен работать в режиме AWE, а при запуске необходимо указать ключ **-h**. Дополнительные сведения об управлении расширениями AWE см. в разделе [Управление памятью для больших баз данных](#).

Допустим, например, что системный администратор работает с SQL Server и Windows Server 2003 выпуска Enterprise Edition на компьютере с 16 ГБ физической памяти. Операционная система позволяет приложениям использовать не более 2 ГБ адресного пространства виртуальной памяти, в SQL Server включен режим AWE, а при запуске был указан параметр **-h**. Чтобы повысить производительность сервера, системный администратор добавляет еще 16 ГБ памяти. SQL Server сразу же распознает дополнительную память и может начать использовать ее сразу по мере необходимости без перезапуска сервера.

#### Примечание

Удаление физической памяти из системы по-прежнему требует перезапуска сервера.

## Основные сведения о неоднородном доступе к памяти

Microsoft SQL Server совместим с архитектурой неоднородного доступа к памяти (NUMA) и хорошо работает на оборудовании NUMA без дополнительной настройки. С ростом тактовой частоты и количества процессоров становится труднее сократить время задержки памяти, необходимой для использования дополнительной производительности системы. Для устранения этого недостатка поставщики оборудования применяют большие кэши третьего уровня, но это является всего лишь полумерой. Архитектура NUMA обеспечивает масштабируемое решение этой проблемы. SQL Server позволяет использовать преимущество компьютеров на основе NUMA без

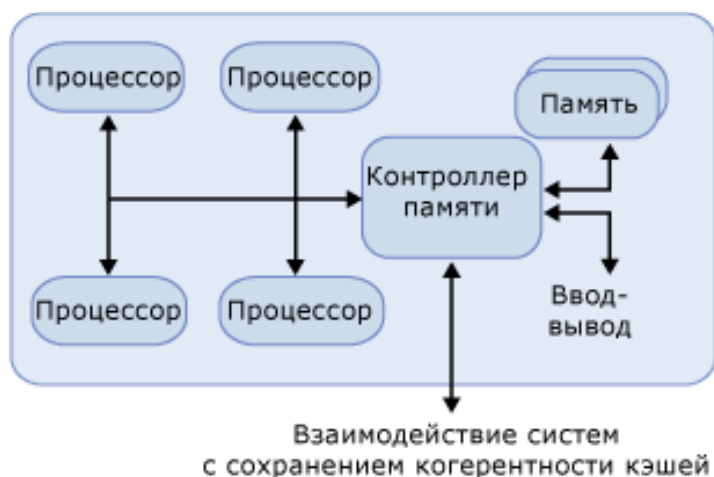
необходимости каких-либо изменений в приложении.

### **Основные понятия NUMA**

Новые тенденции в оборудовании предполагают наличие более одной системной шины, каждая из которых обслуживает небольшое количество процессоров. Каждая группа процессоров имеет свою собственную память и, возможно, свои собственные каналы ввода-вывода. В то же время каждый ЦП может иметь доступ к памяти, связанной с другими группами. Каждая группа называется узлом NUMA. Количество ЦП в узле NUMA определяется поставщиком оборудования. Доступ к локальной памяти происходит быстрее, чем к памяти, связанной с другими узлами NUMA. Поэтому такая структура памяти получила название — архитектура неоднородного доступа к памяти.

На оборудовании NUMA некоторые области памяти физически находятся на шинах, отличных от шин остальных областей. Поскольку NUMA использует локальную и удаленную память, доступ к одним областям производится дольше, чем к другим. Термины локальная память и удаленная память, как правило, употребляются относительно текущего выполняемого потока. Локальная память — это память, которая находится на том же узле, что и ЦП, выполняющий в настоящий момент данный поток. Любая память, не принадлежащая к узлу, на котором в настоящее время выполняется поток, называется удаленной. Удаленная память также иногда называется внешней. Отношение времен доступа к внешней и локальной памяти называется коэффициентом NUMA. Если коэффициент NUMA равен 1, значит, это симметричная многопроцессорная система. Чем выше этот коэффициент, тем больше издержки на доступ к памяти других узлов. Приложения Windows (в том числе SQL Server 2000, пакет обновления 3 и более ранние версии), не поддерживающие архитектуру NUMA, иногда выполняются на оборудовании NUMA неэффективно.

Главный выигрыш от использования NUMA — это масштабируемость. Архитектура NUMA разработана, чтобы преодолеть ограничения масштабирования, присущие архитектуре симметричной мультипроцессорной обработки (SMP). В архитектуре SMP доступ ко всей памяти производится через общую шину памяти. Для относительно небольшого числа процессоров это работает отлично, но при наличии десятков и даже сотен процессоров конкуренция за доступ к шине памяти слишком велика. NUMA устраняет эти узкие места, ограничивая количество ЦП на одной шине памяти и объединяя по высокоскоростному соединению множество узлов.



### *Сравнение оборудования NUMA с программной архитектурой*

NUMA способна уравнивать использование памяти ЦП с помощью специального оборудования (программная архитектура NUMA) или путем настройки памяти SQL Server (программный NUMA). При запуске SQL Server конфигурируется в зависимости от того, на какой операционной системе он выполняется, а также от оборудования или настройки программной архитектуры NUMA. Как для программной архитектуры, так и для оборудования NUMA, при запуске SQL Server в конфигурации NUMA в журнале SQL Server регистрируется мультирежимное конфигурационное сообщение для каждого узла, а также маска ЦП.

### **Оборудование NUMA**

Компьютеры с оборудованием NUMA имеют несколько системных шин, каждая из которых обслуживает небольшую группу процессоров. Каждая группа процессоров имеет свою собственную память и, возможно, свои собственные каналы ввода-вывода. Однако каждый ЦП может иметь доступ к памяти, связанной с другими группами. Каждая группа называется узлом NUMA. Количество ЦП в узле NUMA определяется поставщиком оборудования. Узнать, имеет ли компьютер аппаратную поддержку NUMA, можно у производителя оборудования.

Если имеется оборудование NUMA, оно может быть настроено так, что вместо NUMA используется чередующаяся память. В этом случае Windows и, следовательно, SQL Server, не обнаружат на компьютере аппаратной поддержки NUMA. Выполните следующий запрос, чтобы узнать, сколько узлов памяти доступно SQL Server:

```
SELECT DISTINCT memory_node_id  
FROM sys.dm_os_memory_clerks
```

Если SQL Server возвращает только один узел памяти (узел 0), компьютер либо не имеет оборудования NUMA, либо оборудование настроено на использование чередующейся памяти (не NUMA). Если есть подозрения, что оборудование NUMA настроено неправильно, обратитесь к поставщику оборудования, чтобы он включил поддержку NUMA. SQL Server не обрабатывает конфигурацию NUMA, если в аппаратном режиме NUMA в системе имеется не более четырех ЦП и по крайней мере на одном узле находится только один ЦП.

## Программная архитектура NUMA

SQL Server позволяет группировать ЦП в узлы, называемые программной архитектурой NUMA. Как правило, программная архитектура NUMA конфигурируется, когда имеется несколько ЦП, но нет оборудования NUMA. Программную архитектуру NUMA также можно использовать для дальнейшего разделения NUMA-узлов оборудования на более мелкие группы. Только планировщик SQL Server и сетевой интерфейс SQL Server (SNI) поддерживают программную архитектуру NUMA. Следовательно, программная архитектура NUMA не влияет на узлы памяти, созданные на основе оборудования NUMA. Поэтому, например, если в компьютере с поддержкой SMP установлено 8 ЦП, при создании 4 NUMA-узлов программной архитектуры по 2 ЦП в каждом все четыре NUMA-узла будут обслуживаться лишь одним узлом памяти. Программная архитектура NUMA не обеспечивает соответствия памяти и ЦП.

К преимуществам программной архитектуры NUMA относятся уменьшение количества узких мест операций ввода-вывода и отложенной записи на компьютерах с большим количеством ЦП и без оборудования NUMA. Для каждого NUMA-узла существует один поток ввода-вывода и один поток отложенной записи. В зависимости от использования базы данных эти единичные потоки могут создавать узкие места по производительности. Настройка 4 узлов программной архитектуры NUMA дает четыре потока ввода-вывода и отложенной записи, что может повысить производительность.

Нельзя создать программную архитектуру NUMA, в которую включены ЦП, принадлежащие узлам с разным оборудованием NUMA. Например, если на компьютере установлено оборудование 8 ЦП (0...7) и имеется два узла оборудования NUMA (ЦП 0-3 и 4-7), можно создать программную архитектуру NUMA, объединив ЦП (0,1) и ЦП (2,3). Нельзя создать программную архитектуру NUMA с участием ЦП (1,5), но можно использовать соответствие ЦП, чтобы установить соответствие экземпляра SQL Server с ЦП, принадлежащими различным NUMA-узлам. Для предыдущего примера это означает, что если SQL Server использует ЦП 0-3, для них будет существовать один поток ввода-вывода и отложенной записи. Если в предыдущем примере SQL Server использует ЦП 1, 2, 5 и 6, он получит доступ к двум NUMA-узлам и, соответственно, двум потокам ввода-вывода и отложенной записи.

### Примечание

В некоторых аппаратных конфигурациях общие ресурсы, такие как кэш L3/L4, используются совместно. Вокруг этих общих ресурсов можно группировать процессоры при создании NUMA-узлов программной архитектуры.

Дополнительные сведения см. в разделе [Как настроить сервер SQL Server на использование программной архитектуры NUMA](#).

## Как SQL Server поддерживает архитектуру NUMA

Следующие ключевые изменения, представленные в SQL Server 2005, дают возможность задействовать все преимущества архитектуры неоднородного доступа к памяти (NUMA).



## **Группирование общих ЦП**

SQL Server группирует планировщики для сопоставления с группированием ЦП, основанным на сведениях о границе оборудования NUMA, предоставляемых Windows. Например, 16-процессорный серверный блок может иметь 4 узла NUMA, каждый из которых имеет 4 ЦП. Это позволяет предоставить больше локальной памяти для этой группы планировщиков во время выполнения задач на узле. С SQL Server предоставляется возможность разделения процессоров, связанных с узлом оборудования NUMA, на несколько узлов ЦП. Эта возможность известна как программная архитектура NUMA. Обычно нужно разделить процессоры, чтобы разделить нагрузку между узлами ЦП. Дополнительные сведения о программной архитектуре NUMA см. в разделе [Основные сведения о неоднородном доступе к памяти](#).

Если происходит обработка потока определенным оборудованием NUMA и при этом выделяется память, то диспетчер памяти SQL Server пытается выделить память из памяти, связанной с узлом NUMA для локальных ссылок. Подобным образом страницы буферного пула распределяются между оборудованием NUMA. Для потока более эффективным является доступ к памяти из страницы буфера, которая расположена в локальной памяти, чем доступ из внешней памяти. Дополнительные сведения см. в разделе [Расширение и сжатие буферного пула в конфигурации с неоднородным доступом к памяти \(NUMA\)](#).

Каждый узел NUMA (оборудование или программной архитектуры NUMA) имеет соответствующий порт ввода-вывода, который используется в сетевом дескрипторе ввода-вывода. Это помогает распределять обработку сетевых портов ввода-вывода между несколькими портами. При установлении клиентского соединения с SQL Server происходит установление связи с одним из узлов. Все пакетные запросы от этого клиента будут обрабатываться на этом узле.

Каждый раз, при запуске SQL Server в среде NUMA, в журнал ошибок SQL помещаются информационные сообщения, описывающие конфигурацию NUMA.

## **Как в SQL Server выполняется сопоставление узлов с программной архитектурой NUMA с узлами оборудования NUMA**

Узлы с программной архитектурой NUMA определяются один раз для всех экземпляров SQL Server, содержащихся на компьютере, таким образом, все экземпляры компонента Database Engine будут видеть одни и те же узлы с программным NUMA. Затем каждый экземпляр компонента Database Engine использует [affinity mask option](#), чтобы выбрать нужный ЦП. Каждый экземпляр будет использовать любой узел с программной архитектурой NUMA, который связан с выбранным ЦП.

При запуске Windows распределяет память для операционной системы из аппаратного узла «NODE 0». Соответственно узел «NODE 0» будет иметь меньше локальной памяти, доступной для приложений, чем другие узлы. Данная проблема усложняется, если имеется большой системный кэш-файл. Если SQL Server запускается на компьютере с несколькими узлами NUMA, он попытается запускаться на узле NUMA, отличающемся от узла «NODE 0», чтобы его глобальные структуры могли быть размещены в локальной



памяти. Для настройки программной архитектуры NUMA см. раздел [Как настроить сервер SQL Server на использование программной архитектуры NUMA](#).

### ***Как назначить соединения для узлов NUMA***

Соединения протоколов TCP и VIA могут быть привязаны к одному или нескольким определенным узлам NUMA. Если соединения не привязаны или подключаются через именованные каналы или протокол общей памяти, они распределяются по узлам NUMA по кругу. В пределах узла NUMA соединение выполняется на наименее загруженном планировщике данного узла. Из-за кругового характера назначения новых соединений может быть, что все ЦП узла будут заняты, в то время как другой узел будет свободен. Если количество ЦП очень мало (например 2) и наблюдается большой дисбаланс планирования из-за долго выполняющихся пакетов, например массовой загрузки, возможно, будет получен прирост производительности, если отключить NUMA. Дополнительные сведения см. в разделе [Как сопоставить порты TCP/IP порт с узлами NUMA](#).

### ***Ограничения, налагаемые версиями SQL Server***

В SQL Server 2000 вплоть до версии пакета обновления 3 (SP3) не включена специальная поддержка архитектуры NUMA, однако в пакете обновления 4 (SP4) имеется некоторая ограниченная оптимизация NUMA. В SQL Server 2005 включены существенные улучшения, пользователям NUMA настоятельно рекомендуется выполнить обновление до версии SQL Server 2005, чтобы получить полный спектр преимуществ использования архитектуры NUMA.

## **Расширение и сжатие буферного пула в конфигурации с неоднородным доступом к памяти (NUMA)**

В этом разделе описывается, как распределяются страницы памяти буферного пула в конфигурации с неоднородным доступом к памяти (NUMA). Эти сведения позволяют разобраться, каким образом SQL Server пользуется архитектурой NUMA, и что показывают счетчики объектов буферного пула.

### ***Распределение памяти***

Для каждого физического узла NUMA существует один узел памяти SQL Server. Узлы памяти увеличиваются независимо друг от друга, но память распределяется между ними поровну. Для иллюстрации того, каким образом SQL Server распределяет локальную или внешнюю память, в этом разделе приведен пример, согласно которому на рассматриваемом компьютере установлено 16 ГБ памяти. SQL Server выделяет некоторый объем памяти для собственных процессов за пределами пула буферов, прочим приложениям, включая Windows, требуется некоторый объем памяти из каждого узла, поэтому в распоряжении SQL Server остается 10 ГБ памяти для буферного пула. Память буферного пула распределяется между четырьмя физическими узлами NUMA, N0, N1, N2 и N3 следующим образом:

- N0 — 1 ГБ
- N1 — 3 ГБ
- N2 — 3 ГБ
- N3 — 3 ГБ

В представленной выше конфигурации каждому узлу со временем выделяется 2,5 ГБ памяти, однако расширение узла N0 остановится при достижении 1,0 ГБ собственной памяти, а 1,5 ГБ будут получены от других узлов.

### *Выделение памяти при загрузке*

В архитектуре NUMA SQL Server выделяет память со скоростью, сравнимой с системой без использования NUMA, даже если исходный объем свободной памяти распределен между узлами неравномерно. Буферный пул предпринимает попытку получить максимально возможный объем памяти для каждого узла. Это сложная задача, поскольку у системы Windows нет API-интерфейса для получения памяти от определенного узла.

По мере того как SQL Server расходует память, можно заметить, что некоторые узлы получают большое число страниц от других узлов NUMA (называемых внешними страницами). Однако во время наращивания объема эти страницы не используются, поскольку возможно их частое перемещение к узлу-владельцу и обратно. На момент достижения значения параметра **max server memory** некоторые узлы могут иметь внешнюю память, но как только цель достигнута, буферный пул будет равномерно работать с локальной и со внешней памятью. В частности, при недостатке памяти буферный пул не будет пытаться освободить страницы внешней памяти, пока для этого будут доступны страницы локальной памяти.

### *Ограничение памяти определенными узлами*

Если SQL Server был настроен для работы на подмножестве узлов NUMA, буферный пул не будет автоматически ограничен памятью этих узлов. Для этого следует установить параметр **max server memory**. Сведения о параметре **max server memory** см. в разделе [Параметры памяти сервера](#).

### *Освобождение памяти узла*

В архитектуре NUMA объемы памяти, указываемые параметрами **max server memory** и **min server memory**, равномерно распределяются между узлами. Например, если параметр **max server memory** будет иметь значение 16 ГБ в системе с четырьмя узлами, буферный пул выделит для каждого из них по 4 ГБ памяти. Если один из этих узлов переключить в автономный режим, изменив значение параметра **affinity mask**, то объем, указываемый параметром **max server memory**, будет перераспределен между оставшимися узлами. К примеру, если в ранее рассмотренном случае из четырех узлов в автономный режим перевести два, то освобожденные 8 ГБ памяти будут равномерно распределены между оставшимися узлами. Поскольку буферный пул может использовать внешние страницы, в случае недостаточного объема памяти для оставшихся узлов будет использована удаленная память. Если необходимо, чтобы SQL Server не использовал память узлов, на которых он больше не выполняется, то значение параметра **max server memory** необходимо уменьшить после переключения узлов в автономный режим.

### *Внешние страницы*

Как правило, узлы функционируют независимо друг от друга. Все случаи распределения и освобождения памяти тем или иным узлом реализуются с использованием памяти, связанной с этим узлом. Но если пользователю, работающему на узле N1, потребуется доступ к странице базы данных, расположенной в памяти узла N2, то производится доступ к нелокальной памяти.

### *Наблюдение за использованием локальной и внешней памяти в буферном пуле*

Содержимое буферного пула можно просмотреть с помощью объекта **Buffer Node**. Общий объем памяти буферного пула SQL Server отображается в счетчике Target pages объекта **Buffer Manager**. Объем памяти буферного пула для каждого узла отображается в счетчике Target pages объекта **Buffer Node**. Память, полученная от других узлов, отображается в счетчике Foreign pages. Дополнительные сведения см. в разделах [SQL Server, объект Buffer Node](#) и [SQL Server, объект Buffer Manager](#).

### *Каждый из узлов контролирует свою память*

У каждого узла памяти имеется собственный поток отложенной записи. Он вызывается как для явных, так и неявных контрольных точек. Поскольку у компьютера в симметричной многопроцессорной архитектуре (SMP) имеется всего один поток контрольных точек, несколько потоков при использовании архитектуры NUMA приводят к увеличению их частоты.

### *Поведение при просмотре таблицы*

Оператор просмотра таблицы, выполняемый на узле N1, заполняет только ту память, которая связана с этим узлом (если только просмотр не выполняется параллельно на процессорах нескольких узлов). Если просмотр выполняется исключительно на одном узле, то используются только страницы буфера этого узла. Это помогает секционировать рабочую нагрузку для приложений.

## Сценарии NUMA

На компьютерах с несколькими ЦП оборудование NUMA может значительно повысить производительность, задавая выделенную память для ЦП. В этом разделе описываются некоторые конфигурации NUMA, соответствия процессоров и соединений, которые могут увеличить производительность в определенных случаях. Следующие настройки влияют на эти конфигурации:

- Оборудование NUMA предоставляется производителем компьютера.
- Программная архитектура NUMA настраивается с использованием реестра. Дополнительные сведения о настройке программной архитектуры NUMA см. в разделе [Как настроить сервер SQL Server на использование программной архитектуры NUMA](#).
- Соответствие процессоров настраивается с использованием [параметра affinity mask](#).
- Соответствие портов и NUMA настраивается в качестве параметра сервера с

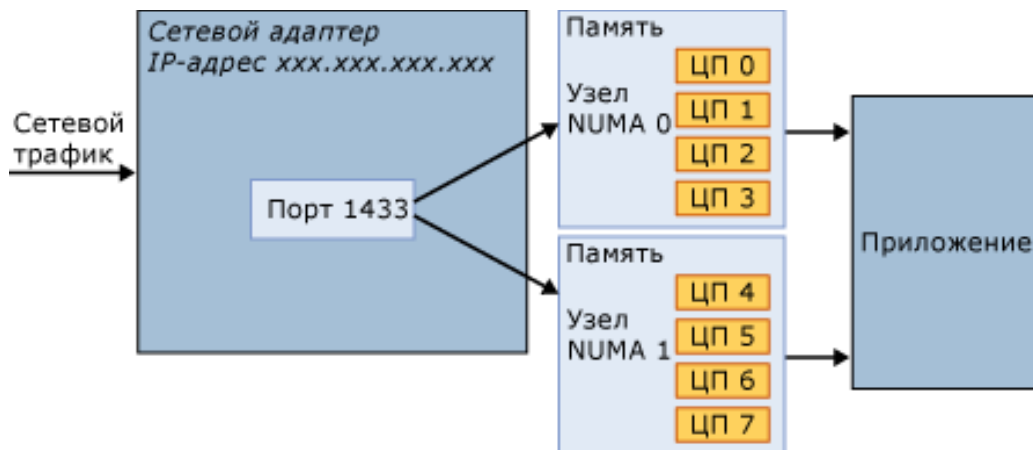
помощью диспетчера конфигурации SQL Server. Дополнительные сведения см. в разделе [Как сопоставить порты TCP/IP порт с узлами NUMA](#).

## Полезные сценарии

При использовании NUMA часто встречаются следующие сценарии.

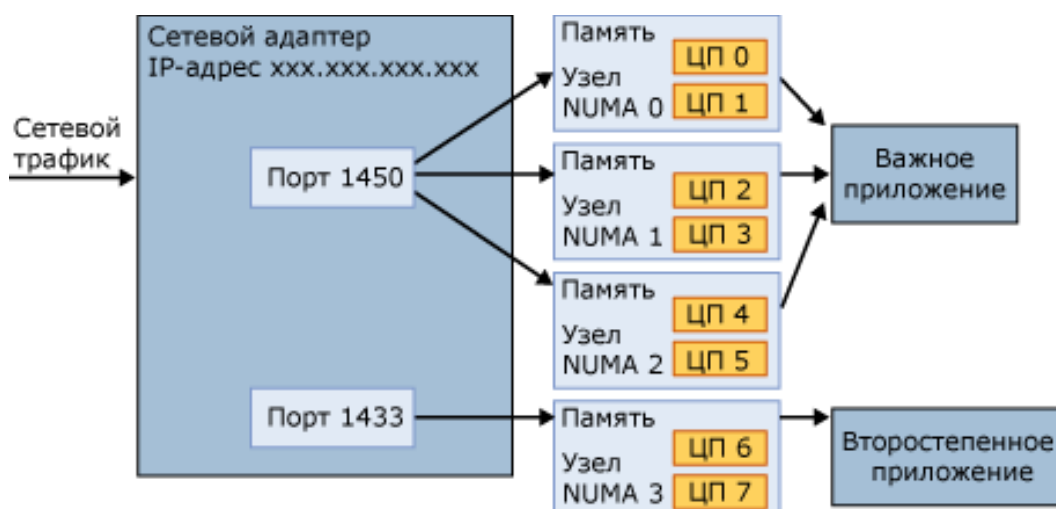
### *А. Нет соответствия портов и NUMA*

Это условие по умолчанию для компьютеров с оборудованием NUMA и единственным экземпляром SQL Server. Весь трафик поступает через один порт и распределяется по всем доступным узлам NUMA циклическим образом. NUMA улучшает размещение памяти и доступ ЦП, увеличивает число потоков ввода-вывода и потоков модуля отложенной записи. Соединения устанавливаются и остаются в данном узле. Это приводит к автоматическому распределению нагрузки по узлам NUMA. Клиентские приложения в таком случае могут подключаться к одному и тому же порту, и поэтому они просты в развертывании.



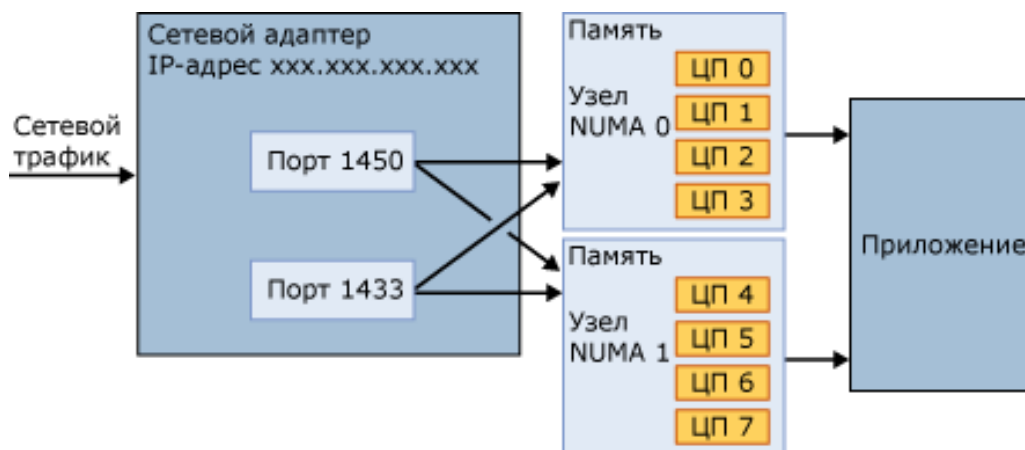
### *Б. Привязка одного порта к нескольким узлам, чтобы обеспечить большую производительность для приоритетного приложения*

Свяжите один порт с несколькими узлами оборудования NUMA, чтобы обслуживать основное приоритетное приложение. Свяжите второй порт с другим узлом NUMA, чтобы обслуживать второе второстепенное приложение. Ресурсы памяти и ЦП распределяются не сбалансировано для двух приложений, обеспечивая главному приложению в три раза больше локальной памяти и ресурсов ЦП, чем второстепенному приложению. Второстепенное приложение может быть вторым экземпляром компонента Database Engine, менее важной функцией того же экземпляра Database Engine или даже той же самой базой данных. Это обеспечивает способ выполнения приоритетного потока, предоставляя дополнительные ресурсы предпочтительному соединению.



### В. Привязка нескольких портов к нескольким узлам

Несколько портов можно сопоставить с одними и теми же узлами NUMA. Это позволяет настроить разные разрешения для разных портов. Например, можно жестко ограничить доступ, предоставляемый через порт, управляя разрешениями на соответствующей конечной точке TCP. В следующем примере к порту 1450 предоставлен широкий доступ через корпоративную сеть. Для доступа к порту 1433 через Internet используется брандмауэр, но доступ жестко ограничен. Оба порта в равной степени используют все достоинства NUMA.



## Архитектура задач и потоков

Потоки — это возможность операционной системы, позволяющая разделить логику приложений на несколько одновременных путей выполнения. Эта особенность полезна, когда сложные приложения имеют много задач, которые могут выполняться одновременно.

Когда операционная система выполняет экземпляр приложения, она создает модуль, называемый процессом, для управления экземпляром. Процесс имеет поток выполнения.

Это ряд программных команд, выполненных кодом приложения. Например, если простое приложение имеет один набор команд, которые могут быть выполнены последовательно, есть только один путь выполнения или поток приложения. Более сложные приложения могут иметь несколько задач, которые могут выполняться одновременно, а не последовательно. Приложение может сделать это, запуская отдельные процессы для каждой задачи. Однако запуск процесса — это ресурсоемкая операция. Вместо этого приложение может запустить отдельные потоки. Они относительно менее ресурсоемки. Кроме этого, каждый поток можно запланировать для выполнения независимо от других потоков, связанных с процессом.

Потоки позволяют сложным приложениям более эффективно использовать ЦП, даже на компьютерах с одним ЦП. С одним ЦП только один поток может выполняться одновременно. Если один поток выполняет длительную операцию, которая не использует ЦП, например считывание с диска или запись на диск, другой поток может выполняться, пока первая операция не завершится. Возможность выполнять потоки, в то время как другие потоки ожидают завершения операции, позволяет приложению увеличить использование ЦП. Это особенно касается многопользовательских приложений, интенсивно использующих операции дискового ввода-вывода, например сервера базы данных. Компьютеры, имеющие несколько микропроцессоров или ЦП, могут выполнять одновременно по одному потоку на каждом ЦП. Например, если компьютер имеет восемь ЦП, он может выполнять одновременно восемь потоков.

## Работа с расписаниями задач и пакетов SQL Server

Каждый экземпляр SQL Server является отдельным процессом операционной системы. Каждый экземпляр должен быть в состоянии обслужить тысячи параллельных запросов пользователей. В экземплярах SQL Server для управления указанными параллельными задачами используются потоки Microsoft Windows либо, в случае наличия соответствующих настроек, волокна. Каждый экземпляр SQL Server всегда задействует несколько потоков системных процессов. К ним относятся один или более потоков, отведенных под сервер Net-Library, сетевой поток управления вводом-выводом, а также сигнальный поток для соединения с диспетчером управления службами.

## Основные сведения о планировании

Каждый экземпляр SQL Server содержит внутренний слой, представляющий собой среду, близкую к операционной системе. Указанный внутренний слой используется для планирования и синхронизации параллельно выполняемых задач без вызова функций ядра Windows. Данный внутренний слой также может быть использован для планирования работы волокон и потоков Windows. Каждый экземпляр SQL Server резервирует для выполнения запросов пользователя пул волокон или потоков Windows. Максимальный размер указанного пула задается серверной настройкой [max worker threads](#).

Для правильного понимания процесса обработки запроса или задачи необходимо ознакомиться с представленными ниже понятиями.



**соединение**

Соединение устанавливается в случае успешной регистрации пользователя. После его установления пользователь может вводить исполняемые инструкции Transact-SQL. Соединение обрывается в случае его удаления или при завершении сеанса пользователем.

**пакет**

Пакет SQL состоит из одной или нескольких инструкций Transact-SQL, отправляемых с клиента на экземпляр SQL Server. Он представляет собой единицу работы, посылаемую пользователем компоненту Database Engine.

**задача**

Задача представляет собой единицу работы, запланированной SQL Server. Пакет может предназначаться одной или несколькими задачами. Например, параллельный запрос выполняется сразу несколькими задачами.

**поток Windows**

Каждый поток Windows представляет собой независимый процесс выполнения задач.

**thread**

thread (Волокно) представляет собой упрощенный поток, для выполнения которого требуется меньшее количество ресурсов Windows и возможно переключение контекстов в пользовательском режиме. Один поток Windows может быть сопоставлен с несколькими волокнами.

**рабочий поток**

Рабочий поток представляет собой логический поток SQL Server, внутренне сопоставленный отношением «один к одному» (1:1) с потоком Windows либо, если включен параметр `lightweight pooling`, с волокном. Указанное сопоставление поддерживается до тех пор, пока рабочий поток не будет освобожден в результате нехватки памяти или превышения интервала ожидания. Взаимосвязь задачи с рабочим потоком поддерживается в течение всего периода выполнения задачи.

## Управление соединениями пользователей и ресурсами рабочих потоков

Даже упрощенные потоки и волокна задействуют достаточное количество ресурсов. В системах с сотнями и тысячами пользовательских соединений выделение одного рабочего потока на каждое соединение может привести к чрезмерной загрузке доступных ресурсов и в результате к снижению производительности SQL Server. Так как большую часть времени соединения находятся в состоянии ожидания пакетов, принимаемых от клиента, то выделение отдельного рабочего потока для каждого соединения является нецелесообразным. Вместо этого экземпляр SQL Server использует пул рабочих потоков. Пул рабочих потоков экземпляра должен иметь объем, достаточный для обслуживания текущего количества параллельных соединений пользователей, выполняющих пакеты. Чтобы эффективно распределить соединения пользователей между несколькими рабочими потоками в экземпляре SQL Server, параметру **max worker threads** необходимо задать значение по умолчанию 0. Это обеспечивает существенное снижение объема используемых ресурсов.

## Настройка SQL Server на использование волокон

Параметр настройки сервера [lightweight pooling](#) определяет использование потоков или волокон Windows в SQL Server. Значением по умолчанию для данного параметра

является 0. При задании данного значения экземпляр SQL Server распределяет потоки Windows между потоками исполнителя согласно значениям, указанным в параметре **max worker threads**. Если параметру **lightweight pooling** присвоено значение 1, то в SQL Server вместо потоков Windows используются волокна. Это называется работой в режиме волокон. В режиме волокон экземпляр SQL Server выделяет один поток Windows для каждого планировщика SQL, а затем для каждого рабочего потока выделяет волокно согласно значениям, указанным в параметре **max worker threads**. В экземпляре SQL Server для планирования и синхронизации задач используются одинаковые алгоритмы как при использовании волокон, так и при работе с потоками Windows. Выпуск SQL Server Express не поддерживает волокна. Дополнительные сведения см. в разделе [Использование параметра lightweight pooling](#). Использовать планирование в режиме волокон для выполнения распространенных операций не рекомендуется. Это может привести к снижению производительности, мешая нормальной работе переключения контекста. Кроме того, некоторые компоненты SQL Server не выполняются правильно в режиме волокон. Дополнительные сведения см. в разделе [использование упрощенных пулов](#).

## Принципы работы с расписаниями задач и пакетов SQL Server

При подключении приложения к компоненту Database Engine ему присваивается идентификатор сеанса (SPID). Вся информация, требующаяся во время сеанса соединения, хранится и обрабатывается во внутренних структурах данных, связанных с идентификатором SPID. По получении экземпляром SQL Server клиентского пакета производится его разбивка на задачи, каждой из которых из пула рабочих потоков выделяется доступный рабочий поток. Привязка рабочего потока к задаче сохраняется в течение всего времени выполнения задачи. Рабочий поток выполняет запросы соответствующего ему планировщика SQL. В том случае, когда все рабочие потоки заняты, но при этом их количество не превышает значение параметра **max worker threads**, экземпляр SQL Server выделяет для нового пакета новый рабочий поток. В случае, когда все рабочие потоки заняты и при этом их количество соответствует значению параметра **max worker threads**, экземпляр SQL Server переводит задачу в режим ожидания до момента высвобождения рабочего потока.

Привязка задачи к рабочему потоку сохраняется до окончательного выполнения указанной задачи, например до окончания отсылки клиенту последнего полученного пакетом результирующего набора. По окончании выполнения предыдущей задачи рабочий поток высвобождается и может быть связан с задачами следующего пакета.

Компонент Database Engine выполняет задания от определенного соединения только в промежутке времени между получением пакета и отправкой результатов клиенту. В течение указанного промежутка времени могут возникнуть ситуации, когда нет необходимости обрабатывать данные пакета. Например, могут появиться промежутки времени, когда компонент Database Engine находится в режиме ожидания окончания получения данных по запросу или высвобождения рабочего потока другим пакетом. Взаимосвязь задачи с рабочим потоком сохраняется даже в случае ее блокирования каким-либо ресурсом.



Когда компонент Database Engine начинает обрабатывать задачу, связанную с пакетом, происходит планирование рабочего потока, связанного с задачей. По окончании выполнения задачи рабочим потоком экземпляр SQL Server перераспределяет его на следующую задачу. В течение сеанса соединения его идентификатор SPID не изменяется. Длительные соединения могут содержать собственные пакетные задачи, выполняемые с помощью нескольких рабочих потоков. Например, задачи первого пакета могут выполняться с помощью рабочего потока worker1, а задачи второго пакета — с помощью worker2. Некоторые инструкции могут выполняться параллельно. В данном случае пакет может содержать несколько задач, одновременно выполняемых несколькими рабочими потоками.

## Распределение потоков ЦП

По умолчанию каждый экземпляр SQL Server запускает один поток. Если функция подобия была включена, операционная система назначает поток конкретному ЦП. Операционная система распределяет потоки экземпляров SQL Server поровну между микропроцессорами или ЦП компьютера в зависимости от загрузки. Иногда операционная система также может перенести поток из загруженного ЦП на другой. Компонент SQL Server Database Engine, в свою очередь, назначает рабочие потоки планировщикам, которые распределяют потоки в равной мере между ЦП.

Параметр **affinity mask** задается при помощи инструкции [ALTER SERVER CONFIGURATION](#). Если параметр **affinity mask** не задан, экземпляр SQL Server распределяет рабочие потоки равномерно между всеми активными планировщиками.

## Использование параметра lightweight pooling

Нагрузка, возникающая при контекстном переключении потоков, не очень высока. Для большинства экземпляров SQL Server установка параметра [lightweight pooling](#) в 0 или 1 не вызовет заметного влияния на производительность. Выиграть от установки параметра **lightweight pooling** могут только те экземпляры SQL Server, которые выполняются там, где:

- установлен мощный многопроцессорный сервер;
- все процессоры работают почти на пределе производительности;
- высок уровень переключения контекста.

В таких системах может наблюдаться небольшое повышение эффективности, если параметр **lightweight pooling** установлен в значение 1.

Использовать планирование в режиме волокон для выполнения обычных операций не рекомендуется. Это может привести к снижению производительности, мешая нормальной работе переключения контекста. Кроме того, некоторые компоненты SQL Server не выполняются правильно в режиме волокон. Дополнительные сведения см. в разделе [Использование упрощенных пулов](#).

## Выполнение потоков и волокон

В Microsoft Windows используется числовая система приоритетов, распределяющая выполняемые потоки по уровням от 1 до 31. Значение 0 зарезервировано для использования операционной системой. При наличии в очереди на выполнение нескольких потоков Windows сначала обслуживает поток с наивысшим приоритетом.

Каждый экземпляр SQL Server по умолчанию имеет уровень приоритета 7, что соответствует стандартному уровню. Это значение по умолчанию задает потокам SQL Server достаточно высокий уровень приоритета, позволяющий им получать достаточно ресурсов ЦП, не снижая производительности других приложений.

Параметр конфигурации [priority boost](#) используется для повышения приоритета потоков экземпляра SQL Server до 13 уровня (наивысший приоритет). Этот параметр назначает потокам SQL Server более высокий приоритет по отношению к другим приложениям. При этом потоки SQL Server всегда будут обслуживаться в первую очередь, а потоки других приложений не смогут занимать ресурсы системы ранее потоков с более высоким приоритетом. Представленные выше меры могут повысить производительность сервера, на котором выполняются только экземпляры SQL Server. Однако если в приложении SQL Server выполняется ресурсоемкая операция, то не рекомендуется присваивать высокий приоритет другим приложениям, так как это может привести к загрузке ресурсов, необходимых для обслуживания потока SQL Server.

Если на компьютере выполняется несколько экземпляров SQL Server, то повышение приоритетов одних может привести к снижению производительности других. Кроме того, включение параметра **priority boost** может привести к простоям других приложений и компонентов, выполняемых на компьютере. Таким образом, данный параметр рекомендуется использовать только в строго определенных условиях.

## ЦП с поддержкой горячей замены

ЦП с поддержкой горячей замены — это возможность динамически добавлять центральные процессоры в запущенную систему. Добавление центральных процессоров может осуществляться физически — путем добавления нового оборудования, логически — путем оперативного аппаратного секционирования, виртуально — через уровень виртуализации. Начиная с SQL Server 2008, SQL Server поддерживает ЦП с поддержкой горячей замены.

Требования для ЦП с поддержкой горячей замены:

- оборудование с поддержкой ЦП с поддержкой горячей замены;
- 64-разрядный выпуск Windows Server 2008 Datacenter или Windows Server 2008 Enterprise Edition для операционных систем на платформе Itanium;
- SQL Server Enterprise.
- SQL Server не может быть настроен на использование программной архитектуры NUMA. Дополнительные сведения о программной архитектуре NUMA см. в разделах [Основные сведения о неоднородном доступе к памяти](#) и [Как настроить сервер SQL Server на использование программной архитектуры NUMA](#).

После добавления центральных процессоров SQL Server не начинает автоматически

использовать их. Это предотвращает использование SQL Server центральных процессоров, добавленных для каких-либо других целей. После добавления центральных процессоров выполните инструкцию [RECONFIGURE](#), что позволит SQL Server распознать новые центральные процессоры в качестве доступных ресурсов.

#### Примечание

Для использования новых центральных процессоров необходимо изменить настройки параметра [affinity64 mask](#).

## Рекомендации по использованию SQL Server на компьютерах, которые имеют более 64 ЦП

В этом разделе представлены рекомендации по использованию экземпляра SQL Server на компьютерах, имеющих более 64 ЦП.

### Связывание аппаратных потоков с процессорами

Не используйте параметры конфигурации сервера `affinity mask` и `affinity64 mask` для привязки процессоров к определенным потокам. Эти параметры поддерживают использование не более 64 ЦП. Используйте вместо этого функцию [ALTER SERVER CONFIGURATION \(Transact-SQL\)](#).

### Управление размером файла журнала транзакций

Не следует полагаться на функцию автоувеличения для увеличения размера файла журнала транзакций. Увеличение журнала транзакций должно происходить последовательно. Увеличение журнала может помешать записи транзакций до тех пор, пока увеличение журнала не будет завершено. Вместо этого заранее выделите место для всех файлов журналов, установив размер файлов в значение, достаточное для поддержки обычной рабочей нагрузки в среде.

## Задание максимальной степени параллелизма для операций с индексами

Производительность операций с индексами, таких как создание или перестроение индекса, можно улучшить на компьютерах с несколькими процессорами, временно используя модель восстановления баз данных с неполным протоколированием или простую модель восстановления. Для таких операций с индексами характерна значительная интенсивность обращения к журналам, а конфликты при обращении к журналу влияют на выбор оптимальной степени параллелизма (DOP), производимый SQL Server.

Кроме того, следует подумать о задании для этих операций параметра максимальной степени параллелизма (MAXDOP). Следующие рекомендации являются общими и основаны на внутренних тестах. Чтобы определить оптимальное значение параметра

MAXDOP для конкретной среды, попробуйте несколько разных значений.

- При использовании модели полного восстановления ограничьте параметр **максимальная степень параллелизма** значением 8 (или меньше).
- При использовании модели восстановления с неполным протоколированием или простой модели восстановления параметру **максимальная степень параллелизма** можно задать значение больше 8.
- Для серверов с настроенным компонентом NUMA параметр MAXDOP не должен превышать количество процессоров, назначенных каждому узлу NUMA. Это связано с тем, что запрос, вероятнее всего, будет использовать локальную память первого узла NUMA, что может снизить время доступа к памяти.
- Для серверов с включенным параметром гиперпоточности, произведенных в 2009 г. или ранее, значение MAXDOP не должно превышать количества физических процессоров.

Дополнительные сведения о параметре **max degree of parallelism** см. в разделе [Степень параллелизма](#).

## Задание максимального числа рабочих потоков

Всегда задавайте максимальное число рабочих потоков таким образом, чтобы оно было больше параметра максимальной степени параллелизма. Количество рабочих потоков всегда должно быть по крайней мере в семь раз больше количества ЦП, имеющихся на сервере. Дополнительные сведения см. в разделе [Параметр max worker threads](#).

## Использование приложений SQL Trace и SQL Server Profiler

В рабочей среде не рекомендуется использовать приложения SQL Trace и Приложение SQL Server Profiler. Издержки использования этих средств также увеличиваются по мере увеличения числа процессоров. Если в рабочей среде необходимо использовать приложение SQL Trace, следует ограничить до минимума число отслеживаемых событий. Следует внимательно тестировать каждое событие отслеживания под нагрузкой и избегать сочетания событий, которые могут значительно повлиять на производительность.

## Задание числа файлов данных базы данных tempdb

Обычно число файлов данных базы данных **tempdb** должно соответствовать количеству процессоров. Однако количество ресурсов для управления базой данных можно уменьшить, внимательно изучив параллельное использование базы данных **tempdb**. Например, если в системе 64 процессора и обычно только 32 запроса используют **tempdb**, увеличение числа файлов **tempdb** до 64 не приведет к повышению производительности. Дополнительные сведения см. в разделе [Оптимизация производительности базы данных tempdb](#).

## Компоненты SQL Server, которые поддерживают более 64 ЦП

В следующей таблице перечислены компоненты SQL Server и указано, поддерживают ли

они более 64 ЦП.

Имя процесса	Исполняемая программа	Использование более 64 ЦП
SQL Server Database Engine	Sqlserver.exe	Да
Службы Reporting Services	Rs.exe	Нет
Службы Analysis Services	As.exe	Нет
Integration Services	Is.exe	Нет
Service Broker	Sb.exe	Нет
Полнотекстовый поиск	Fts.exe	Нет
Агент SQL Server	Sqlagent.exe	Нет
Среда SQL Server Management Studio	Ssms.exe	Нет
Программа установки SQL Server	Setup.exe	Нет

## Планирование аварийного восстановления

Данный раздел содержит сведения о подготовке SQL Server к чрезвычайным ситуациям, что является необходимым для последующего быстрого и эффективного восстановления.

### Подготовка к аварийному восстановлению

При администрировании базы данных SQL Server важна подготовка к возможному аварийному восстановлению. Для аварийного восстановления баз данных необходим правильно созданный и проверенный план резервирования и восстановления SQL Server. Дополнительные сведения см. в разделе [Введение в стратегию резервного копирования и восстановления в SQL Server](#). Кроме того, для гарантии того, что все системы и данные будут быстро восстановлены и включены в работу в форс-мажорных случаях, необходимо создать план аварийного восстановления. При создании плана рассмотрите несколько различных аварийных ситуаций, которые могут повлиять на магазин. Сюда входят природные, например пожар, и технические, такие как одновременный отказ

двух дисков в массиве RAID-5. При создании плана аварийного восстановления определите и подготовьте все необходимые ответные меры для каждого вида аварии. Необходимо протестировать каждый шаг каждого варианта восстановления. Рекомендуется проверять планы аварийного восстановления с помощью симуляции событий, происходящих при авариях.

Создавая план резервного копирования и восстановления, рассматривайте аварийное восстановление в условиях конкретного окружения и бизнес-требований. В качестве примера рассмотрите пожар, произошедший в круглосуточном центре обработки данных. Есть ли уверенность, что можно будет восстановить его работу? Сколько времени пройдет, прежде чем восстановленная система заработает вновь? Какая потеря данных допустима для пользователей?

В идеале план аварийного восстановления должен показывать, сколько времени потребуется на полное восстановление системы и какие данные будут доступны пользователям после его окончания. Например, можно предположить, что после получения необходимого оборудования восстановление потребует 48 часов и данные будут гарантированно восстановлены по состоянию на конец предыдущей недели.

План аварийного восстановления может иметь различную структуру и содержать разные типы данных. План аварийного восстановления может включать:

- план по запросу оборудования;
- план взаимодействия;
- список лиц, с которыми необходимо связаться в случае аварии;
- инструкции по связи с людьми, которые участвуют в ликвидации последствий аварии;
- сведения о том, кто отвечает за выполнение плана;
- список необходимых действий для каждого варианта восстановления. Чтобы оценить прогресс аварийного восстановления, отмечайте начало и завершение выполнения каждой задачи из списка.

## Модели восстановления SQL Server

SQL Server предоставляет три альтернативные модели восстановления: простая модель восстановления, модель полного восстановления и модель восстановления с неполным протоколированием. Модель восстановления является свойством базы данных, которое управляет операциями создания резервных копий и восстановления базы данных. Выбор оптимальной модели восстановления для каждой базы данных является составной частью планирования стратегии резервного копирования и последующего восстановления данных. Выбор модели восстановления для конкретной базы данных зависит от доступности экстендов и от требований к восстановлению. Выбор модели восстановления влияет на вероятность успешного восстановления базы данных после аварии.

Базовые сведения о моделях восстановления см. в разделе [Обзор моделей восстановления](#).

## Управление носителем резервной копии

Рекомендуется включить в план восстановления возможности управления носителем резервной копии, такие как следующие:

- план наблюдения и управления за хранением и утилизацией резервных наборов данных;
- расписание перезаписи носителя резервной копии;
- в многосерверной среде — решение о централизованном или распределенном резервном копировании;
- средства наблюдения за ресурсом носителя;
- процедура для минимизации последствий при утере резервного набора данных или носителя резервной копии (например потеря магнитной ленты);
- решение о хранении резервных наборов данных вне офиса и анализ того, как это повлияет на время восстановления.

Сведения о том, как SQL Server использует устройства резервного копирования и носители, см. в разделе [Работа с носителями резервных копий в SQL Server](#).

## Выполнение сценария базовой функциональности

Обычно сценарий базовой функциональности используется как часть плана аварийного восстановления для подтверждения нормальной работы системы.

Сценарий базовой функциональности предоставляет системному администратору или администратору базы данных надежное средство для проверки того, вернулась ли база данных в работоспособное состояние, избавляя конечных пользователей от необходимости проводить такую проверку.

Сценарий базовой функциональности всегда зависит от конкретного приложения и может иметь различные формы. Например, для системы поддержки принятия решений или системы формирования отчетов таким сценарием могут оказаться всего несколько запросов на формирование ключевых отчетов. Для системы оперативной обработки транзакций (OLTP) этот сценарий может выполнять пакетные хранимые процедуры, которые вызывают инструкции INSERT, UPDATE и DELETE. Например, сценарием базовой функциональности может быть SQL-файл, который посылает серверу пакетные инструкции SQL с помощью программы `sqlcmd`. Другим примером является использование BAT-файла, содержащего команды `bcp` и `sqlcmd`.

## Проверка степени готовности к аварийным ситуациям

Чтобы убедиться в готовности к аварийной ситуации, рекомендуется периодически выполнять следующие действия.

- Проверяйте процедуры резервного копирования и восстановления до того, как произойдет авария. Такая проверка помогает убедиться в наличии всех необходимых резервных копий для восстановления после разных типов аварий, а также в том, что все процедуры четко описаны и могут быть выполнены любым квалифицированным оператором.
- Выполняйте регулярное резервное копирование базы данных и журналов транзакций, чтобы минимизировать потери данных. Корпорация Майкрософт



рекомендует создавать резервные копии как системных, так и пользовательских баз данных.

- Храните системные журналы в безопасном месте. Ведите учет всех устанавливаемых пакетов обновления Microsoft Windows и SQL Server. Ведите учет используемых сетевых библиотек и режима безопасности. Кроме того, если SQL Server использует смешанный режим проверки подлинности (режимы проверки подлинности SQL Server и Windows), запишите пароль **sa** в безопасном месте. Дополнительные сведения см. в разделе [Защита и обеспечение безопасности \(компонент Database Engine\)](#).

#### Важно!

Проверка подлинности Windows намного надежнее, чем проверка подлинности SQL Server. При возможности используйте проверку подлинности Windows.

- Опробуйте шаги, которые будут предприниматься для восстановления после аварии, на другом сервере. При необходимости внесите поправки, требуемые для учета индивидуальных особенностей сервера, и проверьте измененный вариант.
- Используйте сценарий базовой функциональности для быстрой проверки минимальных возможностей системы.

## Аудит и предотвращение потенциально катастрофических ошибок, вызываемых действиями пользователей

Более сложным сценарием восстановления является восстановление после серьезной ошибки, допущенной пользователем, например после случайного удаления объекта базы данных. В этом разделе перечислены средства, которые могут быть полезны при аудите изменений, вносимых в базы данных, а иногда и для контроля за ними.

- Триггеры языка определения данных (DDL).  
Триггеры DDL могут создаваться для аудита или регулирования определенных изменений в схеме базы данных. Триггеры DDL вызывают хранимые процедуры в ответ на ряд инструкций языка DDL. Эти инструкции в основном начинаются ключевыми словами CREATE, ALTER и DROP. Областью действия триггера DDL является либо конкретная база данных, либо весь экземпляр сервера. Дополнительные сведения см. в разделе [Основные сведения о триггерах DDL](#).
- Уведомления о событиях.  
Уведомления о событиях выполняются в ответ на разнообразные инструкции языка определения данных (DDL) Transact-SQL и на события SQL Trace. Уведомления отправляют сведения об этих событиях службе компонента Service Broker.  
Уведомления о событиях могут быть запрограммированы для многих одинаковых событий, зафиксированных SQL Trace. Но, в отличие от создания трассировок, уведомления о событиях могут быть использованы для выполнения действий в ответ на события внутри экземпляра SQL Server. Поскольку уведомления о событиях выполняются асинхронно, эти действия не потребляют ресурсы, определенные немедленной транзакцией. Дополнительные сведения см. в разделе [Уведомления о событиях \(компонент Database Engine\)](#).

#### Примечание

Не все DDL-события можно использовать в триггерах DDL. Некоторые события предназначены только для асинхронных инструкций, не входящих в транзакции. Например, в триггере DDL невозможно использовать событие CREATE



DATABASE. Для таких событий следует применять уведомления о событиях.

- Агент SQL Server.  
Агент — это служба Microsoft Windows, выполняющая запланированные административные задачи, которые называются заданиями. Агент SQL Server использует SQL Server для хранения сведений о заданиях. Помимо прочего, агент SQL Server может выполнять задание в ответ на определенное событие, например в ответ на ошибку с указанной степенью серьезности или номером сообщения. Основные сведения об агенте SQL Server см. в разделе [Автоматизация задач администрирования \(агент SQL Server\)](#). Сведения об использовании агента SQL Server для событий см. в разделе [Наблюдение и обработка событий](#).
- SQL Trace.  
SQL Trace предоставляет системные хранимые процедуры Transact-SQL для создания трассировок выбранных пользователем классов событий в экземпляре SQL Server Database Engine. Эти системные хранимые процедуры можно использовать для создания трассировок вручную в рамках пользовательских приложений. Дополнительные сведения см. в разделе [Знакомство с SQL Trace](#).

#### Примечание

Приложение SQL Server Profiler — это графический пользовательский интерфейс для SQL Trace, с помощью которого можно наблюдать за экземпляром компонента Database Engine или службой Службы Analysis Services. Дополнительные сведения см. в разделе [Работа с приложением SQL Server Profiler](#).

## Восстановление после сбоя

Чтобы восстановить систему после сбоя, необходимо сначала заменить неисправное оборудование, а затем предпринять следующие действия.

1. Установите операционную систему Microsoft Windows, а затем — соответствующий пакет обновления. Проверьте правильность работы домена.
2. Установите компонент Microsoft SQL Server, а затем — соответствующий пакет обновления. Восстановите **главную** базу данных и базу данных **msdb** из резервной копии. Перезагрузите сервер после восстановления **главной** базы данных.

#### Примечание

Если не удастся восстановить **главную** базу данных, то ее необходимо перестроить. Дополнительные сведения см. в разделе [Перестроение системных баз данных](#).

1. Настройте сервер для работы с необходимыми сетевыми библиотеками и режимом безопасности.
2. Проверьте правильность работы экземпляра SQL Server с помощью диспетчера конфигурации SQL Server. Если указанный экземпляр работает некорректно, его необходимо перезапустить. Дополнительные сведения см. в разделе [Как запустить экземпляр SQL Server \(диспетчер конфигурации SQL Server\)](#).
3. Проверьте журнал приложений Windows. В случае если имя компьютера в Windows было изменено, то его необходимо согласовать с именем в SQL Server с помощью хранимых процедур **sp\_dropserver** и **sp\_addserver**. Дополнительные сведения см. в разделах [sp\\_dropserver \(Transact-SQL\)](#) и [sp\\_addlinkedserver \(Transact-SQL\)](#).
4. Восстановите базы данных согласно их индивидуальным планам восстановления.
5. Проверьте доступность системы. Чтобы убедиться в правильности работы, запустите сценарий проверки основных функций.

б.Активируйте возможность работы пользователей в штатном режиме.

## Федеративные серверы баз данных

Для баз данных, которые должны поддерживать системы с высоким уровнем доступности, можно создать федерацию серверов баз данных, что позволит распределить нагрузку на группу серверов, горизонтально секционировав данные базы SQL Server. Эти серверы управляются независимо, но взаимодействуют во время обработки запросов к базе данных.

## Основные сведения о федеративных серверах баз данных

### Проектирование федеративных серверов баз данных

#### Правила планирования федеративных серверов баз данных

Процесс построения федерации серверов баз данных включает в себя разработку набора распределенных секционированных представлений, используемых для распределения данных по серверам. Секционирование дает хорошие результаты, если строение таблиц в базе данных позволяет поделить их на одинаковые секции так, чтобы большинство строк, к которым обращается любая из инструкций SQL, были бы размещены на одном федеративном сервере. Таблицы разбиваются на кластеры связанных элементов. Предположим, например, что записи заказа ссылаются на таблицы **Orders**, **Customers** и **Parts**, а также на все таблицы, которые ведут запись связей между заказчиками, заказами и деталями. Секционирование будет работать наилучшим образом, если все строки в логическом кластере можно будет разместить на одном федеративном сервере.

#### *Симметричные секции*

Наиболее эффективно секционирование работает в случае, если все таблицы в базе данных могут быть секционированы симметрично следующим образом:

- Взаимосвязанные данные размещаются на одном федеративном сервере таким образом, чтобы большинство инструкций SQL, направленных к нужному федеральному серверу, минимально нуждалось в данных, размещенных на других серверах, или вообще не нуждалось в них. Цель проектирования распределенных секционированных представлений можно сформулировать, как правило 80/20: проектируйте секции таким образом, чтобы большинство инструкций SQL обращалось к федеральному серверу, содержащему не менее 80 процентов требуемых данных, а на распределенные запросы приходилось бы 20 или меньше процентов данных. В качестве хорошей проверки достижимости этой цели рекомендуется посмотреть, позволяет ли секция разместить на одном федеративном сервере все строки внешних ключей со всеми их ссылками. Базы данных, структура которых позволяет достичь указанной цели, являются хорошими кандидатами на секционирование.

- Данные секционируются по федеративным серверам равномерно. Предположим, например, что компания поделила Северную Америку на регионы. Каждый работник работает в одном регионе; заказчики совершают большинство покупок в штате или провинции, где живут. Таблицы регионов и работников секционируются между регионами. Данные о заказчиках секционируются между регионами, с учетом штатов или провинций. Хотя некоторые запросы требуют данных из нескольких регионов, большинство запросов нуждается в данных с сервера одного региона. Приложения направляют инструкции SQL на федеративный сервер с данными того региона, который был определен из контекста входных данных.

### **Асимметричные секции**

Хотя идеальными секциями были бы симметричные секции, большинство приложений имеют сложные схемы доступа к данным, не позволяющие осуществить симметричное секционирование. При асимметричном секционировании некоторым федеративным серверам приходится назначать более масштабные роли, чем другим. Например, в базе данных могут быть секционированы лишь некоторые из таблиц, а остальные таблицы, которые не могут быть секционированы, приходится оставлять на исходном сервере. Асимметричное секционирование позволяет добиться большей части производительности, возможной во время симметричного секционирования, и при этом имеет следующие важные преимущества:

- С помощью асимметричного секционирования некоторых таблиц можно значительно улучшить производительность баз данных, которые не могут быть секционированы симметрично.
- Большую существующую систему можно успешно секционировать посредством серии последовательных, асимметричных улучшений. На каждом из шагов для секционирования выбираются таблицы, секционирование которых даст на данном этапе наибольший прирост производительности.

При асимметричном подходе некоторые таблицы, не соответствующие схеме секционирования, как правило, остаются на первоначальном сервере.

Производительность этих оставшихся таблиц обычно более высокая, чем та, что наблюдалась в первоначальной системе, поскольку распределенные таблицы перемещаются на федеративные серверы и, таким образом, сокращается загрузка первоначального сервера.

Многие базы данных могут быть секционированы несколькими способами. Для реализации должны выбираться те конкретные секции, которые наилучшим образом отвечают требованиям типичного диапазона инструкций SQL, выполняемых на уровне бизнес-служб.

## **Проектирование распределенных секционированных представлений**

Чтобы спроектировать набор распределенных секционированных представлений для реализации федерации серверов баз данных, необходимо выполнить следующее:

- Определите шаблон инструкций SQL, выполняемых приложением
- Определите, каким образом таблицы связаны друг с другом.

- Сопоставьте частоту инструкций SQL с секциями, определенными при анализе внешних ключей.
- Определите правила маршрутизации инструкций SQL.

### *Шаблон инструкций SQL, выполняемых приложением*

Создайте список типичных инструкций SQL, которые будут выполняться приложением в течение большей части работы. Разбейте этот список на категории SELECT, UPDATE, INSERT и DELETE и отсортируйте инструкции в каждой из них по частоте выполнения. Если инструкции SQL ссылаются на хранимые процедуры, включайте в список базовые инструкции SELECT, INSERT, UPDATE и DELETE, содержащиеся в этих хранимых процедурах. Если секционируется существующая база данных SQL Server, для получения этого списка можно воспользоваться Приложением SQL Server Profiler.

Упорядочивание инструкций SQL по частоте использования вызвано необходимостью получения разумного их усреднения при типичной оперативной обработке транзакций (OLTP) или работе базы данных веб-узла, в которых применение распределенных секционированных представлений наиболее эффективно. Такие системы характеризуются наличием ряда инструкций SQL, которые извлекают сравнительно небольшие объемы данных по сравнению с запросами, применяемыми в системах принятия решений (или OLAP). Если все инструкции SQL работают с небольшими объемами данных, остается изучить частоту вызова каждой из них, чтобы примерно определить объем данных, проходящих по системе. Однако многие системы содержат некоторое число инструкций SQL, которые запрашивают большое количество данных. Можно также дополнительно оценить то, каким образом влияют запросы, работающие с большими объемами данных.

### *Связи между таблицами*

Целью является найти кластеры таблиц, которые могут быть секционированы по одному измерению (например, по номеру детали или по коду отдела), чтобы все строки, относящиеся к конкретным значениям этого измерения, оказались на одном сервере. Например, может выясниться, что один из способов секционирования базы данных — по региону. Чтобы поддержать эту возможность, даже таблицы, не содержащие в своем ключе кода региона, должны иметь возможность секционирования каким-либо способом, относящимся к региону. Даже если в таблице **Customer** нет столбца кода региона, но регионы определены в виде коллекции штатов и провинций, столбец **Customer.StateProvince** применим для секционирования клиентов по региональному признаку.

Поскольку явные или неявные внешние ключи определяют связи между таблицами, они являются важнейшими элементами при поиске способа секционирования данных. Изучите явные определения внешних ключей, чтобы выяснить, как часто запросы будут пользоваться строками одной таблицы для поиска строк в другой. Кроме того, изучите неявные внешние ключи, а также способы, которыми инструкции SQL пользуются значениями в строках одной таблицы для ссылки на записи в другой при выполнении операций соединения, даже если явные определения ключей отсутствуют. Поскольку

неявные внешние ключи непосредственно в составе схемы базы данных не определены, следует просмотреть инструкции SQL, выдаваемые приложением, чтобы выяснить, используются ли в инструкциях, соединяющих таблицы, неключевые столбцы. Такие неявные внешние ключи обычно индексируются, чтобы повысить производительность операций соединения. Таким образом, следует также просмотреть индексы, определенные в базе данных.

### *Частота использования инструкций SQL применительно к секциям*

Сопоставьте частоту инструкций SQL с секциями, определенными при анализе внешних ключей. Выберите секционирование, которое лучше всего поддерживает весь набор инструкций SQL, используемых в приложении. Если некоторая часть таблиц может быть секционирована более чем одним способом, по частоте использования инструкций SQL определите, какие из секций удовлетворяют большему числу инструкций. Таблицы, которые чаще всего указываются в инструкциях SQL, нужно секционировать в первую очередь. Выстройте последовательность секционирования таблиц, основываясь на частоте ссылок на них.

Шаблон инструкций SQL также влияет на принятие решения о секционировании таблиц:

- Таблица секционируется, если более 5% инструкций, ссылающихся на нее, — это инструкции INSERT, UPDATE или DELETE и если таблица может быть секционирована по выбранному измерению.
- Сохраните полные копии таблиц на каждом из серверов, если менее 5% инструкций, ссылающихся на таблицу, — инструкции INSERT, UPDATE или DELETE. Также необходимо определить, каким образом производятся изменения, чтобы обновлялись все копии таблицы. Если требуется высокая целостность транзакций, можно создать триггеры, выполняющие распределенное обновление всех копий в контексте распределенной транзакции. Если высокая целостность не требуется, можно для переноса изменений от одной копии таблицы на все остальные воспользоваться механизмами репликации SQL Server.
- Не секционируйте и не копируйте таблицу, если менее 5% инструкций, ссылающихся на нее, являются инструкциями INSERT, UPDATE или DELETE, а таблица не может быть секционирована по выбранному измерению.

### *Правила маршрутизации инструкций SQL*

Правила маршрутизации должны быть способны определить, какой из серверов наиболее эффективно обработает каждую из инструкций SQL. Они должны установить связь между контекстом пользовательского ввода и сервером, который содержит данные, необходимые для выполнения инструкции. Приложения должны быть способны принять данные, введенные пользователем, и сопоставить их с правилами маршрутизации, чтобы определить, какой из серверов должен обработать данную инструкцию SQL.

## **Проектирование федеративных серверов баз данных для обеспечения высокого уровня доступности**

Данные для больших веб-узлов или внутренних систем оперативной обработки транзакций (OLTP) должны иметь высокую степень доступности. При использовании

кластера на уровне приложения потеря одного сервера может снизить производительность системы, но не останавливает всю систему. Оставшиеся серверы в кластере производят повторную балансировку нагрузки, пока в кластер не будет включен заменяющий сервер.

Хотя SQL Server не поддерживает создание кластера с балансированием нагрузки, есть возможность использовать отказоустойчивый кластер служб Microsoft Cluster Services. Отказоустойчивый кластер поддерживает от одного до четырех серверов на кластер, в зависимости от операционной системы. Кластер представляется приложениям как один виртуальный сервер. При выходе из строя узла основного сервера другой узел обнаруживает потерю основного сервера и автоматически начинает обслуживание всех запросов, посылаемых виртуальному серверу. Кластер продолжает работать с альтернативным узлом до тех пор, пока основной сервер не будет починен или заменен. Отказоустойчивый кластер способствует обеспечению высокого уровня доступности, но не производит никакой балансировки нагрузки.

## Реализация федеративных серверов баз данных

### Создание распределенных секционированных представлений

Прежде чем реализовывать секционированные представления, необходимо расположить первую секцию горизонтально. При конструировании секционированных схем должно быть понятно, какие данные принадлежат каждой таблице-элементу. Исходная таблица заменяется несколькими меньшими таблицами-элементами. Каждая таблица-элемент и исходная таблица имеют одинаковое количество столбцов, у каждого столбца такие же атрибуты, как и у соответствующего ему столбца в исходной таблице (тип данных, размер, параметры сортировки). При создании распределенного секционированного представления каждая таблица-элемент находится на отдельном сервере. Чтобы расположения были как можно прозрачнее, имена баз данных-элементов должны быть одинаковыми на каждом сервере-элементе, хотя это и не обязательно.

Например: **Server1.CustomerDB**, **Server2.CustomerDB**, **Server3.CustomerDB**.

#### *Создание таблиц-элементов*

Таблицы-элементы конструируются так, чтобы в каждой таблице хранился горизонтальный фрагмент исходной таблицы, основанный на интервале значений ключа. Интервалы основаны на значениях данных в столбце секционирования. Диапазон значений в каждой таблице-элементе определяется проверочным ограничением на столбце секционирования, и интервалы не пересекаются. Например, нельзя, чтобы в одной таблице был интервал от 1 до 200000, а в другой — от 15000 до 300000, иначе будет неясно, какая таблица содержит значения от 15000 до 200000.

Например, таблица **Customer** секционируется на три таблицы. Для этих таблиц будет задано следующее проверочное ограничение:

```
-- On Server1:
```

```
CREATE TABLE Customers_33
  (CustomerID    INTEGER PRIMARY KEY
    CHECK (CustomerID BETWEEN 1 AND 32999),
  ... -- Additional column definitions)

-- On Server2:
CREATE TABLE Customers_66
  (CustomerID    INTEGER PRIMARY KEY
    CHECK (CustomerID BETWEEN 33000 AND 65999),
  ... -- Additional column definitions)

-- On Server3:
CREATE TABLE Customers_99
  (CustomerID    INTEGER PRIMARY KEY
    CHECK (CustomerID BETWEEN 66000 AND 99999),
  ... -- Additional column definitions)
```

### *Определение распределенных секционированных представлений*

После создания таблиц-элементов на каждом сервере-элементе определяется распределенное секционированное представление, при этом у всех представлений будет одно и то же имя. Это позволяет запросам, которые ссылаются на имя распределенного секционированного представления, запускать один или несколько серверов-элементов. Система действует так, как если бы на каждом сервере-элементе была копия исходной таблицы, но у каждого сервера есть только одна таблица-элемент и одно распределенное секционированное представление. Расположение данных прозрачно для приложения.

При построении распределенных секционированных представлений выполняются следующие задачи:

- Добавление определений связанных серверов на каждый сервер-элемент, содержащий сведения о соединении, необходимые для запуска распределенных запросов на других серверах-элементах. Это предоставляет распределенному секционированному представлению доступ к данным на других серверах.
- Установка параметра **lazy schema validation** для всех определений связанных серверов, применяемых в распределенных секционированных представлениях, с помощью процедуры **sp\_serveroption**. Это оптимизирует производительность, поскольку исполнитель запросов не запрашивает метаданные ни для одной из связанных таблиц до тех пор, пока это действительно не понадобится удаленной таблице-элементу.
- Создание распределенного секционированного представления на всех серверах-элементах. Представления обращаются к данным из связанных серверов-элементов с помощью распределенных инструкций **SELECT** и соединяют распределенные строки со строками из локальной таблицы-элемента.

Чтобы создать распределенное секционированное представление для предыдущего примера, выполните следующие действия:

- Добавьте определение связанного сервера под именем **Server2** с данными о соединении для **Server2**, а также определение связанного сервера под именем **Server3** для доступа к **Server3**.
- Создайте следующее распределенное секционированное представление:

```
CREATE VIEW Customers AS
  SELECT * FROM CompanyDatabase.TableOwner.Customers_33
```

- ```
UNION ALL
  SELECT * FROM Server2.CompanyDatabase.TableOwner.Customers_66
UNION ALL
  SELECT * FROM Server3.CompanyDatabase.TableOwner.Customers_99
```
- Выполните те же действия на Server2 и Server3.

### **Правила таблиц**

Таблицы-элементы определяются в предложении FROM в каждой инструкции SELECT в определении представления. Каждая таблица-элемент должна соответствовать следующим правилам:

- На таблицы-элементы нельзя ссылаться в представлении больше одного раза.
- У таблиц-элементов не может быть индексов, созданных на любых вычисляемых столбцах.
- Все ограничения первичного ключа таблиц-элементов должны быть на одинаковом количестве столбцов.
- У таблиц-элементов должна быть одна и та же настройка дополнения ANSI. Дополнительные сведения о настройке дополнения ANSI см. в разделе [SET ANSI\\_PADDING](#).

### **Правила столбцов**

Столбцы определяются в списке выборки каждой инструкции SELECT в определении представления. Каждый столбец должен соответствовать следующим правилам.

- Все столбцы в каждой таблице-элементе должны быть включены в список выборки. Допустимый синтаксис: SELECT \* FROM <таблица-элемент>.
- На столбцы нельзя ссылаться в списке выборки больше одного раза.
- В списке выборки у столбцов должен быть такой же порядок следования.
- Тип столбцов в списке выборки каждой инструкции SELECT должен совпадать. Это относится к типам данных, точности, масштабу и параметрам сортировки. Например, следующее определение представления будет ошибочным, поскольку первый столбец в обоих выражениях SELECT имеет разный тип данных:

```
CREATE VIEW NonUpdatable
AS
SELECT IntPrimaryKey, IntPartNmbr
FROM FirstTable
  UNION ALL
SELECT NumericPrimaryKey, IntPartNmbr
FROM SecondTable
```

### **Правила столбцов секционирования**

Для секционирования может использоваться только один столбец, и он должен присутствовать в каждой таблице-элементе. Проверочные ограничения определяют данные, доступные в каждой таблице-элементе. Применяются следующие дополнительные правила:

- Диапазон ключа проверочных ограничений в таблице не должен пересекаться с диапазонами любой другой таблицы. Любое значение столбца секционирования должно быть сопоставленным только с одной таблицей. Проверочные ограничения могут использовать только следующие операторы: BETWEEN, IN,



AND, OR, <, <=, >, >=, =.

- Столбец секционирования не может быть столбцом с автоматическим приращением, столбцом по умолчанию или столбцом типа **timestamp**.
- Столбец секционирования должен находиться в одинаковом порядковом расположении в списках выборки всех инструкций SELECT в представлении. Например, в каждом списке выборки столбец секционирования всегда является первым, вторым и т.д.
- Столбец секционирования не может содержать значения NULL.
- Столбец секционирования должен быть частью первичного ключа таблицы.
- Столбец секционирования не может быть вычисляемым столбцом.
- У столбца секционирования может быть только одно ограничение. Если задано несколько ограничений, то SQL Server игнорирует их все и не учитывает их при выяснении того, является ли это представление секционированным.
- Ограничения для обновления столбца секционирования отсутствуют.

Столбец секционирования, соответствующий всем этим правилам, будет поддерживать все оптимизации, которые поддерживаются оптимизатором запросов. Дополнительные сведения см. в разделе [Разрешение распределенных секционированных представлений](#).

## Общие правила

### Примечание

Следующие условия не применяются к локально секционированным представлениям на том же сервере. Эта функция включена для обратной совместимости.

Ниже приведены некоторые дополнительные правила:

- Распределенное секционированное представление нельзя сформировать с помощью операторов EXCEPT или INTERSECT.
- Чтобы обеспечить атомность среди всех узлов, которые затрагивает обновление, будет запущена распределенная транзакция.
- Параметр XACT\_ABORT SET должен иметь значение ON.
- Столбцы **smallmoney** и **smalldatetime** в удаленных таблицах отображаются как **money** и **datetime** соответственно. Следовательно, соответствующие столбцы в локальных таблицах должны также иметь имена **money** и **datetime**.
- Любой связанный сервер не может быть замкнут на себя. Это связанный сервер, указывающий на тот же экземпляр SQL Server.

Представление, которое ссылается на секционированные таблицы, но не соответствует всем этим правилам, может поддерживать обновление, если в представлении имеется триггер INSTEAD OF. Однако планы выполнения для представлений с триггером INSTEAD OF, построенные оптимизатором запросов, могут не всегда быть настолько же эффективными, как планы для секционированных представлений, соответствующих всем правилам.

## Изменение данных в секционированных представлениях

Если секционированное представление не подлежит обновлению, оно может служить лишь копией исходной таблицы, предназначенной только для чтения. Обновляемое секционированное представление обладает всеми возможностями исходной таблицы.

Представление считается обновляемым секционированным, если оно представляет

собой набор инструкций SELECT, результаты которых объединяются в один набор инструкцией UNION ALL. Каждая инструкция SELECT ссылается на одну базовую таблицу SQL Server. Эта таблица может быть локальной или связанной таблицей, на которую ссылается полное имя из четырех частей, функция OPENROWSET или функция OPENDATASOURCE (функции OPENROWSET или OPENDATASOURCE, определяющие передаваемый запрос, использовать нельзя).

Кроме того, инструкции, изменяющие данные и ссылающиеся на представление, должны следовать правилам, определенным для инструкций INSERT, UPDATE и DELETE.

#### Примечание

Массовый импорт в секционированное представление не поддерживают ни команда **bcp**, ни инструкции BULK INSERT и INSERT ... SELECT \* FROM OPENROWSET(BULK...). Однако можно вставить в секционированное представление несколько строк с помощью инструкции [INSERT](#).

Если секционированное представление распределено по серверам, избегайте использования триггеров и каскадных действий в базовых таблицах. Триггер или каскадное действие могут внести изменения в базовые данные, влияющие на определение представления.

Распределенные секционированные представления могут быть обновлены, если пользователь имеет разрешения CONTROL, ALTER, TAKE OWNERSHIP или VIEW DEFINITION на каждую базовую таблицу представления. Дополнительные сведения см. в разделе [Устранение неполадок с видимостью метаданных в распределенных секционированных представлениях](#).

#### Примечание

Изменять данные через распределенные секционированные представления можно, только если установлен выпуск SQL Server 2008 Enterprise или SQL Server 2008 Developer. Но изменять данные через локальные секционированные представления можно в любом выпуске SQL Server 2008.

### Инструкции INSERT

Инструкции INSERT добавляют данные в базовые таблицы через секционированное представление. Инструкции INSERT должны следовать следующим правилам.

- В инструкцию INSERT должны включаться все столбцы, даже если в базовой таблице значением столбца может быть NULL или если для него определено ограничение DEFAULT.
- Ключевое слово DEFAULT в предложении VALUES инструкции INSERT использовать нельзя.
- В инструкциях INSERT должны содержаться значения, удовлетворяющие логике ограничения CHECK, определенного для секционированного столбца одной или более базовых таблиц.
- Инструкции INSERT запрещены, если в одной из базовых таблиц есть столбец удостоверений.
- Инструкции INSERT запрещены, если одна из базовых таблиц содержит столбец типа timestamp.
- Инструкции INSERT запрещены, если есть самосоединение с тем же самым

представлением или с одной из базовых таблиц.

### **Инструкции UPDATE**

Инструкции UPDATE изменяют данные в одной или более базовых таблицах через секционированное представление. Инструкции UPDATE должны следовать следующим правилам:

- Ключевое слово DEFAULT в предложении SET инструкции UPDATE использовать нельзя, даже если в соответствующей базовой таблице для данного столбца определено значение DEFAULT.
- Значение столбца удостоверений не может быть изменено; однако другие столбцы обновлять можно.
- Значение столбца с ограничением PRIMARY KEY нельзя изменить, если он содержит данные типа text, image или ntext.
- Инструкции UPDATE запрещены, если в одной из базовых таблиц есть столбец типа timestamp.
- Инструкции UPDATE запрещены, если есть самосоединение с тем же самым представлением или с одной из базовых таблиц.

### **Инструкции DELETE**

Инструкции DELETE удаляют данные в одной или более базовых таблицах через секционированное представление. Инструкции DELETE запрещены, если есть самосоединение с тем же самым представлением или с одной из базовых таблиц.

## **Резервное копирование и восстановление федеративных серверов баз данных**

На уровне федеративных серверов баз данных, который создается при помощи распределенных секционированных представлений, серверы участников формируют один логический блок. Следовательно, необходимо координировать восстановление баз данных участников, чтобы обеспечить их правильную синхронизацию.

SQL Server не требует осуществления координации резервных копий на серверах участников. Резервные копии могут быть взяты независимо из каждой базы данных, без учета состояния баз данных другого участника. Так как синхронизация резервных копий не обязательна, нет затрат на синхронизацию и нет блокирования текущих задач.

Наиболее важный аспект восстановления ряда баз данных участника такой же, как и при восстановлении любой другой базы данных: Планирование и проверка процедур восстановления до введения базы данных в эксплуатацию. Необходимо установить один логический момент времени для процессов восстановления всех баз данных. SQL Server предоставляет возможности фиксации одного момента времени для восстановления всех баз данных участника.

# Таблицы

## Основные сведения о таблицах

### Основы таблиц

### Основы целостности данных

Первым этапом указания доменов таблицы является определение типов данных столбцов. Домен — это множество всех допустимых для столбца значений. Это понятие включает в себя не только контроль типов данных, но и набор значений, допустимых для столбца. Например, домен столбца **Color** таблицы **Production.Product** включает в себя тип данных `nvarchar` и ограничение размера до 15 символов. Домен может также определять символьные строки, допустимые для столбца, например: «красный», «синий», «зеленый», «желтый», «коричневый», «черный», «белый», «тиковый», «серый» и «серебристый». Дополнительные сведения см. в разделе [Типы данных \(компонент Database Engine\)](#).

#### *Значения NULL*

Значения NULL могут приниматься или отвергаться в качестве значений столбца. NULL — это специальное значение в базах данных, представляющее собой понятие неизвестного значения. Значение NULL отличается от символа пробела или значения 0. В действительности пробел является допустимым символом, а значение 0 — допустимым числом. Значение NULL означает, что неизвестно, что собой представляет данное значение. Оно отличается также и от строки нулевой длины. Если определение столбца содержит предложение `NOT NULL`, в него нельзя вставлять строки, имеющие значение NULL для этого столбца. Если определение столбца содержит ключевое слово `NULL`, значения NULL в этом столбце допустимы.

Если для столбца разрешены значения NULL, это может повысить сложность логических сравнений, в которых участвуют значения этого столбца. Стандарт ISO устанавливает, что результатом сравнения со значением NULL может быть не `TRUE` или `FALSE`, а `UNKNOWN`. Это вводит использование в операторах сравнения трехзначной логики, которую трудно реализовать правильно.

#### *Ограничения, правила, значения по умолчанию и триггеры*

Кроме типа данных и размера, столбцы таблицы имеют и другие свойства, которые являются важной частью механизма обеспечения целостности данных и ссылочной целостности таблиц в базе данных:

- Целостность данных означает, что все вхождения столбца должны иметь допустимые значения. Данные должны иметь верный тип данных и правильный домен.
- Ссылочная целостность означает правильность поддержки связей между таблицами. Данные, содержащиеся в одной таблице, должны указывать только на существующие строки в другой таблице.

Для обеспечения целостности обоих типов используются следующие объекты:

- Ограничения
- Правила
- Значения по умолчанию
- Триггеры DML

### Ограничения

Ограничения позволяют задать метод, с помощью которого компонент Database Engine автоматически обеспечивает целостность базы данных. Ограничения задают правила допустимости определенных значений в столбцах и представляют собой стандартный механизм обеспечения целостности. Предпочтительнее использовать ограничения, а не [Триггеры DML](#), [правила](#) и [значения по умолчанию](#). Оптимизатор запросов также использует определения ограничений для построения высокопроизводительных планов выполнения запросов.

### Классы ограничений

SQL Server поддерживает следующие классы ограничений:

- Ограничение NOT NULL указывает, что в столбце недопустимы значения NULL. Дополнительные сведения см. в разделе [Разрешение значений NULL](#).
- Ограничения CHECK обеспечивают целостность домена путем ограничения значений, которые могут быть помещены в столбец. Дополнительные сведения см. в разделе [Ограничения CHECK](#).  
Ограничение CHECK задает логическое условие поиска (принимая значение TRUE, FALSE или unknown), которое применяется ко всем значениям, вставляемым в столбец. Все значения, для которых получается значение FALSE, отбрасываются. Для каждого столбца можно указать несколько ограничений CHECK. В следующем примере показано создание ограничения chk\_id. Это ограничение, среди прочего, следит за соблюдением домена первичного ключа, обеспечивая, чтобы в качестве значений ключа вводились только числа указанного диапазона.  

```
CREATE TABLE cust_sample  
(  
    cust_id          int      PRIMARY KEY,  
    cust_name        char(50),  
    cust_address      char(50),  
    cust_credit_limit money,  
    CONSTRAINT chk_id CHECK (cust_id BETWEEN 0 and 10000 )  
)
```
- Ограничения UNIQUE обеспечивают уникальность значений в наборе столбцов. Ограничение UNIQUE не допускает использования одного и того же значения в двух строках столбца. Первичные ключи также обеспечивают уникальность, однако не допускают использования значения NULL в качестве одного из уникальных значений. Дополнительные сведения см. в разделе [Ограничения UNIQUE](#).
- Ограничения PRIMARY KEY используются для указания столбца или набора столбцов, которые имеют значения, уникально идентифицирующие строку в таблице. Дополнительные сведения см. в разделе [Ограничения PRIMARY KEY](#). Две строки таблицы не могут иметь одинаковых значений первичного ключа. В столбцах первичного ключа недопустимо значение NULL. В качестве первичного

ключа рекомендуется использовать небольшой столбец с целочисленными значениями. Каждая таблица должна иметь первичный ключ. Столбец или сочетание столбцов, которые соответствуют требованиям значений первичного ключа, называются потенциальными ключами.

В следующем примере создается таблица `part_sample` с указанием поля `part_nmbr` в качестве первичного ключа.

```
CREATE TABLE part_sample
    (part_nmbr      int      PRIMARY KEY,
     part_name      char(30),
     part_weight    decimal(6,2),
     part_color     char(15) );
```

- Ограничения `FOREIGN KEY` задают и обеспечивают связи между таблицами. Дополнительные сведения см. в разделе [Ограничения FOREIGN KEY](#). Внешний ключ одной таблицы указывает на потенциальный ключ в другой таблице. В следующем примере в таблице `order_part` создается внешний ключ, который ссылается на таблицу `part_sample`, определенную ранее.

```
CREATE TABLE order_part
    (order_nmbr     int,
     part_nmbr      int
      FOREIGN KEY REFERENCES part_sample(part_nmbr)
      ON DELETE NO ACTION,
     qty_ordered    int);
GO
```

Вставить строку со значением внешнего ключа, для которого отсутствует потенциальный ключ с таким значением, нельзя, за исключением строки со значением `NULL`. Предложение `ON DELETE` определяет действия, предпринимаемые при попытке удаления строки, на которую указывает существующий внешний ключ. В предложении `ON DELETE` предусмотрены следующие параметры:

- `NO ACTION`: удаления не произойдет и будет выведено сообщение об ошибке.
- `CASCADE`: все строки с внешними ключами, указывающими на удаленную строку, также будут удалены.
- `SET NULL`: всем строкам с внешними ключами, указывающими на удаленную строку, присваивается значение `NULL`.
- `SET DEFAULT`: всем строкам с внешними ключами, указывающим на удаленную строку, присваивается установленное для них значение по умолчанию. Дополнительные сведения см. в разделе [Умолчания](#).

Предложение `ON UPDATE` определяет действия, предпринимаемые при попытке обновления значения потенциального ключа, на которое указывает существующий внешний ключ. Это предложение также поддерживает параметры `NO ACTION`, `CASCADE`, `SET NULL` и `SET DEFAULT`.

## Ограничения столбцов и таблиц

Ограничения могут относиться к столбцам или к таблицам. Ограничение столбца указывается в описании столбца и применяется только к данному столбцу. Ограничения, показанные в предыдущих примерах, являются ограничениями столбцов. Ограничение таблицы объявляется независимо от описаний столбцов и может быть применено к нескольким столбцам таблицы. Ограничения таблиц используются при необходимости включить в ограничение нескольких столбцов.

Например, если таблица имеет два или более столбцов в первичном ключе, для включения в ограничение всех столбцов первичного ключа необходимо использовать ограничение таблицы. Представьте таблицу, в которую записываются все события, происходящие в заводском компьютере. Предположим, что события нескольких типов могут происходить одновременно, однако в одно и то же время не может произойти двух событий одного типа. Этого можно достичь, если включить в первичный ключ таблицы, состоящий из двух столбцов, и столбец event\_type, и столбец event\_time, как показано в следующем примере.

```
CREATE TABLE factory_process
(event_type    int,
 event_time    datetime,
 event_site    char(50),
 event_desc    char(1024),
CONSTRAINT event_key PRIMARY KEY (event_type, event_time) )
```

### **Правила**

Правила — это средства обеспечения обратной совместимости, которые по функциональности напоминают ограничения CHECK. Применение ограничений CHECK является предпочтительным стандартным способом ограничения диапазона значений столбца. Кроме того, ограничения CHECK более лаконичны, чем правила. Со столбцом может быть связано только одно правило, а ограничений CHECK несколько. Ограничения CHECK указываются в инструкции CREATE TABLE, а правила создаются как отдельные объекты, которые потом связываются со столбцами.

#### **Важно!**

В будущей версии Microsoft SQL Server эта возможность будет удалена. Избегайте использования этой возможности в новых разработках и запланируйте изменение существующих приложений, в которых она применяется. Используйте вместо этого ограничения CHECK. Дополнительные сведения см. в разделе [Ограничения](#).

В следующем примере создается правило, позволяющее записывать в столбец только те числа, которые попадают в указанный диапазон. После этого правило связывается со столбцом таблицы.

```
CREATE RULE id_chk AS @id BETWEEN 0 and 10000;
GO
CREATE TABLE cust_sample
(
  cust_id          int
  PRIMARY KEY,
  cust_name        char(50),
  cust_address     char(50),
  cust_credit_limit money,
);
GO
sp_bindrule id_chk, 'cust_sample.cust_id';
GO
```

### **Умолчания**

Значения по умолчанию определяют, какими значениями заполнять столбец, если при вставке строки для этого столбца значение не указано. Значение по умолчанию могут



быть любым выражением, результат которого — константа, например собственно константой, встроенной функцией или математическим выражением.

Чтобы использовать значения по умолчанию, создайте их определения с помощью ключевого слова DEFAULT инструкции CREATE TABLE. При этом константное выражение будет присвоено столбцу как значение по умолчанию. Дополнительные сведения см. в разделе [Изменение и создание определений DEFAULT](#).

В следующем примере создается таблица, в которой используются различные типы выражений по умолчанию. Здесь объект по умолчанию, назначенный столбцу, создается и связывается со столбцом. Затем производится тестовая вставка данных без указания значений для столбцов со значениями по умолчанию, а также выборка этой тестовой строки для проверки того, что были использованы все значения по умолчанию.

```
USE AdventureWorks2008R2;
GO
CREATE TABLE test_defaults
    (keycol          smallint,
     process_id      smallint DEFAULT @@SPID,    --Preferred default definition
     date_ins        datetime DEFAULT getdate(), --Preferred default definition
     mathcol         smallint DEFAULT 10 * 2,    --Preferred default definition
     char1           char(3),
     char2           char(3) DEFAULT 'xyz') --Preferred default definition;
GO
/* For illustration only, use DEFAULT definitions instead.*/
CREATE DEFAULT abc_const AS 'abc';
GO
sp_bindefault abc_const, 'test_defaults.char1';
GO
INSERT INTO test_defaults(keycol) VALUES (1);
GO
SELECT * FROM test_defaults;
GO
```

Ниже приводится результирующий набор.

Default bound to column.

(1 row(s) affected)

| keycol | process_id | date_ins           | mathcol | char1 | char2 |
|--------|------------|--------------------|---------|-------|-------|
| 1      | 7          | Oct 16 1997 8:34PM | 20      | abc   | xyz   |

(1 row(s) affected)

## Типы специальных таблиц

Кроме стандартной роли обычных определяемых пользователем таблиц SQL Server предоставляет в базах данных следующие типы таблиц специального назначения:

- Секционированные таблицы
- Временные таблицы
- Системные таблицы
- Широкие таблицы



### **Секционированные таблицы**

Секционированные таблицы — это таблицы, данные которых горизонтально разделены на блоки, которые могут быть распределены между несколькими файловыми группами в базе данных. Секционирование делает большие таблицы и индексы более управляемыми, позволяет быстро и эффективно получать доступ к наборам данных и управлять ими, при этом сохраняя целостность всей коллекции. Согласно сценарию секционирования такие операции, как загрузка данных из систем OLTP или OLAP, могут занимать секунды вместо минут или часов в предыдущих выпусках. Операции обслуживания, выполняемые на наборах данных, также производятся значительно эффективнее, так как нацелены только на те данные, которые действительно необходимы, а не на всю таблицу.

Секционировать таблицу имеет смысл, если она очень велика или может значительно разрастись и выполняется любое из следующих условий:

- Таблица содержит или будет содержать большое число данных, использующихся разными способами.
- Запросы к таблице или ее обновления выполняются не так, как ожидается, а затраты на обслуживание превышают заранее определенные сроки.

Секционированные таблицы обладают всеми свойствами и особенностями, относящимися к проектированию стандартных таблиц и созданию к ним запросов, в том числе ограничениями, значениями по умолчанию, значениями идентификаторов и временных меток, триггерами и индексами. Следовательно, если нужно реализовать секционированное представление на локальном сервере, вместо этого следует создать секционированную таблицу. Дополнительные сведения, которые помогут в понимании, проектировании и реализации секционированных таблиц, см. в разделе [Секционированные таблицы и индексы](#).

### **Временные таблицы**

Существует два вида временных таблиц: локальные и глобальные. Локальные временные таблицы видны только их создателям до завершения сеанса соединения с экземпляром SQL Server, как только они впервые созданы или когда на них появляется ссылка. Локальные временные таблицы удаляются после отключения пользователя от экземпляра SQL Server. Глобальные временные таблицы видны всем пользователям в течение любых сеансов соединения после создания этих таблиц и удаляются, когда все пользователи, ссылающиеся на эти таблицы, отключаются от экземпляра SQL Server.

Дополнительные сведения о временных таблицах см. в разделе [Основы создания и изменение таблиц](#).

### **Системные таблицы**

SQL Server хранит данные, определяющие конфигурацию сервера и всех его таблиц в специальном наборе таблиц, называемых системными. Пользователи не могут напрямую обращаться к системным таблицам или обновлять их, кроме случаев использования выделенного административного соединения (DAC), которое должно выполняться только под контролем службы сопровождения клиентов Microsoft.

Дополнительные сведения см. в разделе [Использование выделенного административного соединения](#). Обычно системные таблицы меняются с каждой новой версией SQL Server. Приложения, которые напрямую ссылаются на системные таблицы, возможно, необходимо будет переписать перед переходом на новый выпуск SQL Server с другими версиями системных таблиц. Данные из системных таблиц доступны через представления каталога. Дополнительные сведения см. в разделе [Системные таблицы \(Transact-SQL\)](#).

#### **Важно!**

Системные таблицы компонента SQL Server 2008 Database Engine реализованы как представления только для чтения, что обеспечивает обратную совместимость с SQL Server 2008. Напрямую работать с данными этих системных таблиц нельзя. Рекомендуется осуществлять доступ к метаданным SQL Server через [представления каталога](#).

### **Широкие таблицы**

Широкая таблица — это таблица, в которой определен набор столбцов. В широких таблицах используются [разреженные столбцы](#), что позволяет увеличить общее количество столбцов в таблице до 30 000. Количество индексов и статистики также увеличивается до 1000 и 30 000 соответственно. Максимальный размер строки широкой таблицы — 8 019 байт. Таким образом, большинство данных в любой строке должны составлять значения NULL. Чтобы создать или преобразовать обычную таблицу в широкую, следует добавить в определение таблицы [набор столбцов](#). Максимальным числом неразряженных и вычисляемых столбцов в широкой таблице остается 1024.

С помощью широких таблиц можно создавать в приложении гибкие схемы. Добавлять или удалять столбцы можно в любой момент. Следует учитывать, что у широких таблиц существуют особенности производительности, например увеличенное время выполнения и требования к памяти во время компиляции.

### **Вопросы производительности широких таблиц**

Широкая таблица — это таблица с набором столбцов. При использовании широких таблиц следует учитывать следующие вопросы производительности.

- В широких таблицах может быть определено до 30 000 столбцов. Это увеличивает затраты на обслуживание индексов для таблицы. Определяемые некластеризованные индексы должны быть отфильтрованными индексами, применяющимися к подмножествам данных. Дополнительные сведения см. в разделе [Рекомендации по проектированию отфильтрованных индексов](#).
- Приложения могут динамически добавлять или удалять столбцы из широких таблиц. При добавлении или удалении столбцов скомпилированные планы запросов также становятся недействительными. Рекомендуется разрабатывать приложение с учетом предполагаемой рабочей нагрузки, чтобы изменения в схеме были сведены к минимуму.
- Добавление или удаление данных из широкой таблицы может влиять на производительность. Приложения следует разрабатывать с учетом предполагаемой рабочей нагрузки, чтобы изменения в схеме были сведены к минимуму.

- Рекомендуется ограничить количество индексов широкой таблицы только теми, которые необходимы для бизнес-логики. С ростом числа индексов возрастает время компиляции DML и требования к памяти.
- В широкой таблице следует ограничить выполнение инструкций DML, обновляющих несколько строк ключа кластеризации. Для компиляции и выполнения этих инструкций может потребоваться значительный объем ресурсов.
- Операции по переключению секций в широких таблицах могут выполняться медленно. Для их выполнения может потребоваться значительный объем памяти. Требования к производительности и памяти пропорциональны общему количеству столбцов в исходной и целевой секциях.
- Курсоры обновления, обновляющие определенные столбцы в широкой таблице, должны явно перечислять столбцы в предложении FOR UPDATE. Это поможет оптимизировать производительность при использовании курсоров.

## Конструирование таблиц

### Правила планирования таблиц

При проектировании базы данных прежде всего нужно решить, какие таблицы понадобятся, какого типа данные будут храниться в каждой из них, а также какие пользователи будут иметь к ним доступ. Перед созданием таблицы и связанных с ней объектов рекомендуется наметить в общих чертах планы и принять решения по следующим характеристикам таблиц.

- Типы данных, которые будут храниться в таблице.
- Число столбцов в таблице и тип данных и длина (если это необходимо) каждого столбца.
- Какие столбцы могут принимать значения NULL.
- Использовать ли ограничения, значения по умолчанию и правила; если да, то где.
- Типы индексов, которые понадобятся, какие столбцы делать первичными ключами, а какие — внешними.

Самый эффективный способ создать таблицу — определить все, что от нее нужно сразу же. Сюда входят ограничения на данные и дополнительные компоненты. После создания таблиц и работы с ними можно продолжить более детальную их разработку.

При создании таблиц полезно начать с простой таблицы, добавить в нее данные и некоторое время поработать с ней. Этот подход дает возможность определить, транзакции какого типа выполняются чаще всего, и какого типа данные чаще всего вводятся, до того, как дизайн таблицы будет зафиксирован ограничениями, индексами, значениями по умолчанию, правилами и другими объектами.

### Присвоение типа данных столбцу

Присвоение типа данных каждому столбцу является одним из первых шагов, предпринимаемых при проектировании таблицы. Типы данных определяют допустимые значения данных для каждого столбца. Типы данных могут присваиваться столбцам одним из следующих способов.

- С помощью системных типов данных SQL Server.

- Созданием псевдонимов типов данных, основанных на системных типах данных.
- Созданием определяемых пользователем типов на базе типов, созданных в среде CLR Microsoft .NET Framework.

Например, если в столбец должны включаться только имена, столбцу может быть присвоен один из символьных типов данных. Аналогично, если столбец должен содержать только числа, можно присвоить ему один из числовых типов данных. Дополнительные сведения о типах данных см. в разделе [Типы данных \(компонент Database Engine\)](#).

SQL Server поддерживает также синонимы SQL-92 для нескольких базовых типов данных. Дополнительные сведения см. в разделе [Синонимы типов данных \(Transact-SQL\)](#).

### **Принудительное обеспечение целостности данных**

Системные, определяемые пользователем типы данных, а также синонимы типов данных могут использоваться для принудительного обеспечения целостности данных. Это является следствием того, что вводимые или изменяемые данные должны соответствовать типу, заданному в исходной инструкции CREATE TABLE. Например, нельзя сохранить фамилию в столбце, определенном как datetime, так как в столбце типа datetime могут вводиться только допустимые значения дат. Вообще, числовые данные должны содержаться в числовых столбцах, особенно если в дальнейшем должны выполняться вычисления на основе этих данных.

### **Данные в строке**

Типы больших значений размером от небольшого до среднего (varchar(max), nvarchar(max), varbinary(max) и xml) и типы данных LOB (text, ntext и image) могут храниться в строке данных. Это управляется двумя параметрами в системной хранимой процедуре **sp\_tableoption**: параметром **LARGE VALUE TYPES OUT OF ROW** для типа больших значений и параметром **TEXT IN ROW** для типов больших объектов. Их лучше всего использовать для таблиц, в которых значения любого из вышеуказанных типов данных обычно считываются и записываются в одной единице измерения, и большинство инструкций, содержащих ссылки на таблицу, обращаются к этому типу данных. В зависимости от характеристик интенсивности использования и рабочей нагрузки хранение данных в строках может быть неоптимальным.

#### **Важно!**

Параметр **TEXT IN ROW** не будет использоваться в следующей версии SQL Server. Не рекомендуется использовать его в новых разработках и постараться внести изменения в приложения, которые в настоящее время используют этот параметр. Кроме того, следует хранить объемные данные в типах varchar(max), nvarchar(max) или varbinary(max). Для управления режимом в строке и вне строки этих типов данных используйте параметр **LARGE VALUE TYPES OUT OF ROW**.

Если параметру **TEXT IN ROW** не присвоено значение ON или конкретное ограничение «в строке», строки text, ntext и image являются объемными символьными или двоичными строками (до 2 ГБ), хранимыми вне строки данных. Строка данных

содержит только 16-байтный текстовый указатель на корневой узел дерева, составленного из внутренних указателей. Эти указатели задают соответствие со страницами, в которых хранятся фрагменты строк. Дополнительные сведения о хранении строк text, ntext и image см. в разделе [Использование типов данных text и image](#).

Предусмотрена возможность задания параметра **TEXT IN ROW** для таблиц, содержащих столбцы типов данных LOB. Также можно задать предельное значение для параметра **TEXT IN ROW** в диапазоне от 24 до 7 000 байт.

Аналогично, если параметру **LARGE VALUE TYPES OUT OF ROW** не присвоено значение ON, столбцы varchar(max), nvarchar(max), varbinary(max) и xml при возможности хранятся в строке данных. В этом случае компонент SQL Server Database Engine пытается подобрать конкретное значение, если это возможно; в противном случае он принудительно выносит значения за пределы строки. Если параметру **LARGE VALUE TYPES OUT OF ROW** присвоено значение ON, значения хранятся вне строки, а в записи хранится только 16-байтный текстовый указатель.

#### Примечание

Предельным объемом для хранения больших типов-значений является 8 000 байт, если параметр **LARGE VALUE TYPES OUT OF ROW** имеет значение OFF. В отличие от параметра **TEXT IN ROW**, для столбцов таблицы нельзя задавать ограничение «в строке».

Когда таблица настроена для хранения типа больших значений или LOB непосредственно в строке данных, фактические значения столбцов находятся в строке, если выполняется любое из следующих условий.

- Длина строки меньше ограничения, заданного для столбцов text, ntext и image.
- В строке данных достаточно места для хранения строки.

Если значение столбца с типом больших значений или LOB хранится в строке данных, компонент Database Engine не должен обращаться к отдельной странице или набору страниц для считывания или записи в символьную или двоичную строку. Это делает операции считывания и записи в строки «в строке» практически такой же операцией, что и считывание и запись в строки varchar, nvarchar или varbinary ограниченного размера. Аналогично, если значения хранятся вне строки, компонент Database Engine выполняет дополнительное считывание или запись в страницу.

Если строка не превышает предельное значение параметра **TEXT IN ROW** или доступное пространство в строке, то для типов данных LOB набор указателей, в других случаях хранимых в корневом узле дерева указателей, записывается в строку. Указатели записываются в строку, если выполняется любое из следующих условий:

- объем дискового пространства, требуемого для хранения указателей, меньше заданного в параметре **TEXT IN ROW** предельного значения;
- в строке данных достаточно места для хранения указателей.

Если указатели перемещены из корневого узла в саму строку, компонент Database Engine не обязательно должен использовать корневой узел. Это позволяет избежать необходимости доступа к страницам при считывании или записи в строку и тем самым повысить производительность.

Если используются корневые узлы, они хранятся в качестве одного из фрагментов строки в странице LOB и могут содержать до пяти внутренних указателей. Компоненту Database Engine требуется 72 байта в строке для хранения пяти указателей строки «в строке». Если в строке недостаточно места для хранения указателей и параметр **TEXT IN ROW** имеет значение ON или параметр **LARGE VALUE TYPES OUT OF ROW** имеет значение OFF, компоненту Database Engine, возможно, потребуется выделить страницу объемом 8 КБ для их хранения. Если длина данных значения превышает 40 200 байт, необходимо более пяти указателей «в строке», при этом только 24 байта хранятся в основной строке, и выделяется дополнительная страница данных в пространстве для хранения LOB.

Объемные строки хранятся в строке аналогично строкам переменной длины. Компонент Database Engine сортирует столбцы по убыванию размера и выносит значения за пределы строки до тех пор, пока оставшиеся столбцы не будут уместиться в странице данных (8 КБ).

### Включение и отключение параметра **large value types out of row**

Параметр **LARGE VALUE TYPES OUT OF ROW** для таблицы можно включить хранимой процедурой **sp\_tableoption** следующим образом:

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

При указании значения OFF ограничитель длины строк для столбцов **varchar(max)**, **nvarchar(max)**, **varbinary(max)** и **xml** устанавливается в значение в 8 000 байт. В строке хранится только 16-байтный корневой указатель, а само значение содержится в пространстве для хранения LOB. Рекомендуем присваивать этому параметру значение ON для таблиц, в которых большинство инструкций не ссылается на столбцы большого типа-значения. Внестрочное хранение этих столбцов подразумевает, что можно уместить большее количество строк на страницу, тем самым уменьшая количество операций ввода-вывода, затрачиваемых на сканирование таблицы.

Если этому параметру присвоено значение OFF, многие строки могут в итоге храниться в самой строке, потенциально уменьшая количество строк данных, помещающихся в каждой странице. Если большинство инструкций, содержащих ссылки на таблицу, не обращаются к столбцам типа **varchar(max)**, **nvarchar(max)**, **varbinary(max)** и **xml**, уменьшение количества строк на странице может привести к увеличению количества страниц, которые необходимо прочитать при обработке запросов. Уменьшение количества строк на страницу потенциально может увеличить количество страниц, которые должны быть отсканированы, если оптимизатор не найдет подходящего индекса.

Также можно использовать хранимую процедуру **sp\_tableoption** для отключения параметра **OUT OF ROW**:

```
sp_tableoption N'MyTable', 'large value types out of row', 'OFF'
```

При изменении значения параметра **LARGE VALUE TYPES OUT OF ROW** текущие значения типа **varchar(max)**, **nvarchar(max)**, **varbinary(max)** и **xml** не будут

преобразованы моментально. Порядок хранения строк меняется при их дальнейшем обновлении. Все новые значения, вставленные в таблицу, сохраняются в соответствии с действующим параметром таблицы.

Чтобы проверить значение параметра **LARGE VALUE TYPES OUT OF ROW** для конкретной таблицы, выполните запрос к столбцу **large\_value\_types\_out\_of\_row** представления каталога **sys.tables**. Значение этого столбца равно 0, если в таблице не включен параметр **LARGE VALUE TYPES OUT OF ROW**, и 1, если типы больших значений хранятся вне строк.

### Включение и отключение параметра **TEXT IN ROW**

Параметр **TEXT IN ROW** для таблицы можно включить хранимой процедурой **sp\_tableoption** следующим образом:

```
sp_tableoption N'MyTable', 'text in row', 'ON'
```

При необходимости можно задать предельное значение в диапазоне от 24 до 7 000 байт для длины строки **text**, **ntext** и **image**, которую можно хранить в следующей строке данных:

```
sp_tableoption N'MyTable', 'text in row', '1000'
```

Если вместо конкретного предела будет указано значение **ON**, пределом по умолчанию будет 256 байт. Это значение по умолчанию позволяет использовать большую часть преимуществ по производительности, предоставляемых параметром **TEXT IN ROW**. Хотя обычно не требуется устанавливать значение меньше 72, слишком большое значение также не рекомендуется использовать. Особенно это важно для таблиц, в которых большинство инструкций не содержат ссылки на столбцы **text**, **ntext** и **image** или в которых содержится несколько столбцов **text**, **ntext** и **image**.

Если задано большое ограничение на значение параметра **TEXT IN ROW** и в самой строке хранится много строк, можно существенно уменьшить количество строк данных, помещающихся в каждой странице. Если большинство инструкций, содержащих ссылки на таблицу, не обращаются к столбцам **text**, **ntext** или **image**, уменьшение количества строк в странице может увеличить количество страниц, которые должны быть считаны при обработке запросов. Уменьшение количества строк на страницу потенциально может увеличить размер индексов и страниц, которые должны быть отсканированы, если оптимизатор не найдет подходящего индекса. Значения по умолчанию в 256 байт для ограничения **TEXT IN ROW** достаточно, чтобы обеспечить хранение небольших строк и корневых текстовых указателей в строках, но недостаточно для такого снижения количества строк на страницу, чтобы повлиять на производительность.

Параметру **TEXT IN ROW** автоматически присваивается значение 256 байт для переменных типа **table** и таблиц, возвращаемых пользовательскими функциями. Это значение невозможно изменить.

Также можно использовать хранимую процедуру **sp\_tableoption** для отключения этого

параметра, задав в ней в качестве его значения **OFF** или **0**:

```
sp_tableoption N'MyTable', 'text in row', 'OFF'
```

Чтобы проверить значение параметра **TEXT IN ROW** для конкретной таблицы, выполните запрос к столбцу **text\_in\_row\_limit** представления каталога **sys.tables**. Значение этого столбца равно 0, если параметр **TEXT IN ROW** для таблицы отключен, и больше 0, если задано ограничение «в строке».

### **Результаты использования параметра TEXT IN ROW**

Параметр **TEXT IN ROW** оказывает следующее влияние.

- После включения параметра **TEXT IN ROW** можно применять инструкции **TEXTPTR**, **READTEXT**, **UPDATETEXT** и **WRITETEXT** для считывания и редактирования частей любого значения **text**, **ntext** или **image**, сохраненного в таблице. В инструкциях **SELECT** можно считывать всю строку **text**, **ntext** или **image** или использовать функцию **SUBSTRING** для считывания частей строки. Все инструкции **INSERT** и **UPDATE**, которые содержат ссылки на таблицу, должны задавать полные строки и не могут вносить изменения в часть строки **text**, **ntext** или **image**.
- При первом включении параметра **TEXT IN ROW** существующие строки **text**, **ntext** и **image** не преобразуются сразу в строки «в строке». Это происходит только в случае их последующего обновления. Все строки **text**, **ntext** и **image** после включения параметра **TEXT IN ROW** вставляются как строки «в строке».
- Отключение параметра **TEXT IN ROW** может быть ресурсоемкой регистрируемой в журнале операцией. При этом блокируется таблица и все строки «в строке» **text**, **ntext** и **image** преобразуются в обычные строки **text**, **ntext** и **image**. Продолжительность выполнения команды и объем редактируемых данных зависит от того, сколько строк **text**, **ntext** и **image** необходимо преобразовать из строк «в строке» в обычные строки.
- Параметр **TEXT IN ROW** не оказывает влияния на работу ODBC-драйвера собственного клиента SQL Server или поставщика OLE DB для собственного клиента SQL Server, за исключением ускорения доступа к данным типа **text**, **ntext** и **image**.
- Функции работы с типами данных **text** и **image** из библиотеки DB-Library, такие как **dbreadtext** и **dbwritetext**, нельзя использовать для таблицы после включения параметра **TEXT IN ROW**.

### **Превышающие размер страницы данные строки, превышающие 8 КБ**

Строка таблицы может содержать максимум 8 060 байт. В SQL Server 2008 это ограничение сделано менее строгим в отношении таблиц, содержащих столбцы **varchar**, **nvarchar**, **varbinary**, **sql\_variant** или столбцы определяемого пользователем типа данных CLR. Длина каждого из этих столбцов по-прежнему не может быть больше 8000 байт; но их общая длина может превышать предел 8060 байт. Это применимо в отношении столбцов **varchar**, **nvarchar**, **varbinary**, **sql\_variant** или столбцов определяемого пользователем типа данных CLR при их создании или изменении, а также при обновлении и вставке данных.



**Примечание**

Данное ограничение не относится к столбцам `varchar(max)`, `nvarchar(max)`, `varbinary(max)`, `text`, `image` или `xml`.  
Дополнительные сведения о хранении этих столбцов см. в разделах [Использование типов данных больших значений](#), [Использование типов данных `text` и `image`](#) и [Использование XML-данных](#).

**Замечания относительно переполнения строк**

При объединении столбцов `varchar`, `nvarchar`, `varbinary`, `sql_variant` или столбцов определяемого пользователем типа данных CLR, длина которых превышает 8 060 байт на строку, необходимо учитывать следующее:

- Превышение предела в 8 060 байт на строку может повлиять на производительность, поскольку SQL Server по-прежнему поддерживает максимум 8 КБ на страницу. Если сочетание столбцов `varchar`, `nvarchar`, `varbinary`, `sql_variant` или столбцов определяемого пользователем типа данных CLR выходит за этот предел, компонент SQL Server Database Engine перемещает столбец записи с наибольшей шириной на другую страницу в единице распределения `ROW_OVERFLOW_DATA`, оставляя при этом 24-разрядный указатель на исходной странице. Перемещение больших записей на другую страницу осуществляется динамически, по мере удлинения записей при операциях обновления. Операции обновления, которые укорачивают записи, могут привести к возвращению записей на исходную страницу в единице распределения `IN_ROW_DATA`. Кроме того, выполнение запросов и других операций выборки, например сортировки и соединения, в отношении больших записей с превышающими размер страницы данными строки, увеличивает время обработки, поскольку эти записи обрабатываются синхронно, а не асинхронно. Поэтому при построении таблиц с несколькими столбцами `varchar`, `nvarchar`, `varbinary`, `sql_variant` или столбцами определяемого пользователем типа данных CLR следует учитывать процент строк, в которых возможно переполнение и вероятную частоту запросов данных переполнения. Если ожидаются частые запросы по многим превышающим размер страницы данным строки, рекомендуется нормализовать таблицу таким образом, чтобы некоторые столбцы переместились в другую таблицу. После этого запросы по таблице можно будет выполнять с помощью асинхронной операции JOIN.
- Длина отдельных столбцов `varchar`, `nvarchar`, `varbinary`, `sql_variant` и столбцов определяемого пользователем типа данных CLR по-прежнему не должна превышать 8 000 байт. И только общая их длина может выходить за предел в 8 060 байт на строку таблицы.
- Сумма столбцов данных других типов, включая `char` и `nchar`, не должна превышать 8 060 байт на строку. Данные больших объектов также могут выходить за предел в 8 060 байт на строку.
- Ключ индекса кластеризованного индекса не может содержать столбцы типа `varchar`, в которых имеются данные в единице распределения `ROW_OVERFLOW_DATA`. Если кластеризованный индекс создается для столбца типа `varchar` и существующие данные располагаются в единице распределения `IN_ROW_DATA`, то все последующие операции вставки или обновления для данного столбца, выталкивающие данные за пределы строки, будут завершаться ошибкой. Дополнительные сведения о единицах распределения см. в разделе [Организация таблиц и индексов](#).
- Пользователь может включить столбцы, которые содержат превышающие размер страницы данные строки, в качестве ключевых или неключевых столбцов

- некластеризованного индекса.
- Максимальный размер записи в таблицах, в которых используются разреженные столбцы, составляет 8 018 байт. Если суммарная величина преобразуемых данных и существующих данных записи превышает 8 018 байт, то возвращается ошибка MSSQLSERVER ERROR 576. Если выполняется преобразование разреженных и неразреженных типов, то компонент Database Engine хранит копию текущих данных записи. В связи с этим удваивается количества места, которое требуется для хранения записи.
- Для получения сведений о таблицах или индексах, которые могут содержать превышающие размер страницы данные строки, используется функция динамического управления [sys.dm\\_db\\_index\\_physical\\_stats](#).

## Автоматическая нумерация и столбцы идентификаторов

Для каждой таблицы может быть задан единственный идентификатор столбца, который содержит сформированные системой последовательные значения, уникально определяющие каждую строку в таблице. Например, при вставке строк в таблицу столбец идентификаторов может автоматически сформировать для приложения уникальные номера квитанций заказчиков. Как правило, столбцы идентификаторов содержат значения, уникальные в пределах таблицы, в которой они определены. Это значит, что в других таблицах, содержащих столбцы идентификаторов, могут использоваться такие же значения. Однако обычно это не приводит к ошибкам, поскольку значения идентификаторов используются чаще всего в пределах одной таблицы, а столбцы идентификаторов различных таблиц не связаны между собой.

Единый, глобально уникальный столбец идентификаторов можно создать для таблицы, которая содержит значения, уникальные для всех сетевых компьютеров в мире. Столбец, который содержит гарантированно глобально уникальные значения, часто бывает нужен при слиянии аналогичных данных из нескольких систем баз данных. Например, он может понадобиться в пользовательской биллинговой системе, данные которой находятся в различных подразделениях компании по всему миру. При слиянии данных на центральном веб-узле для консолидации и создания отчетов благодаря глобально уникальным значениям покупателя в различных странах или регионах никогда не получат одинаковые биллинговые номера или идентификаторы заказчиков.

При репликации слиянием или транзакционной репликации с обновлением подписок SQL Server обеспечивает уникальность идентификации строк в нескольких копиях таблицы с помощью столбцов идентификаторов GUID.

## Вычисляемые столбцы

Вычисляемый столбец вычисляется на основе выражения, в котором могут использоваться другие столбцы той же таблицы. Выражение может быть именем невычисляемого столбца, константой, функцией или любым их сочетанием, соединенным одним или несколькими операторами. Выражение не может быть вложенным запросом.

Например, в образце базы данных База данных AdventureWorks2008R2 определение столбца **TotalDue** таблицы **Sales.SalesOrderHeader** следующее: **TotalDue AS Subtotal +**

**TaxAmt + Freight.**

Если не указано иное, вычисляемые столбцы являются виртуальными столбцами, которые физически в таблице не хранятся. Их значения вычисляются заново каждый раз при обращении к ним запроса. В компоненте Database Engine для физического хранения вычисляемых столбцов в таблицах используется ключевое слово **PERSISTED** инструкций **CREATE TABLE** и **ALTER TABLE**. Значения столбцов обновляются каждый раз при изменении любых столбцов, входящих в вычисляемое выражение. Пометив вычисляемый столбец как **PERSISTED**, можно создать на вычисляемом столбце индекс, являющийся детерминистическим, но неточным. Кроме того, если вычисляемый столбец обращается к функции среды CLR, компонент Database Engine не может определить, является ли эта функция подлинно детерминистической. В этом случае вычисляемый столбец необходимо пометить как **PERSISTED**, чтобы на нем можно было создавать индексы. Дополнительные сведения см. в разделе [Создание индексов вычисляемых столбцов](#).

**Примечание**

Все вычисляемые столбцы, используемые в столбцах секционирования секционированной таблицы, должны быть явно помечены как **PERSISTED**.

Вычисляемые столбцы могут использоваться в списках выборки, предложениях **WHERE**, **ORDER BY** и в любых других местах, в которых могут использоваться обычные выражения, за исключением следующих случаев.

- Вычисляемые столбцы, используемые в качестве ограничений **CHECK**, **FOREIGN KEY** или **NOT NULL**, должны быть помечены как **PERSISTED**. Вычисляемый столбец может использоваться в качестве ключевого столбца в индексе в качестве компонента какого-либо ограничения **PRIMARY KEY** или **UNIQUE**, если значение этого вычисляемого столбца определено детерминистическим выражением, а тип данных результата разрешен в столбцах индекса. Например, если в таблице есть целочисленные столбцы **a** и **b**, вычисляемый столбец **a + b** можно проиндексировать, а вычисляемый столбец **a + DATEPART(dd, GETDATE())** проиндексировать нельзя, так как его значение может измениться при последующих вызовах.
- Вычисляемый столбец не может быть целевым столбцом инструкций **INSERT** или **UPDATE**.

Компонент Database Engine автоматически определяет возможность хранения значения **NULL** вычисляемыми столбцами на основе используемых выражений. Результат большинства выражений может быть равен **NULL** даже в случае, если присутствуют только столбцы, не допускающие **NULL**, так как при возможных переполнениях или потерях точности будут выданы результаты с этим значением. Для выяснения возможности хранения значения **NULL** вычисляемым столбцом в таблице используется функция **COLUMNPROPERTY** со свойством **AllowsNull**. Выражение, допускающее **NULL**, можно превратить в не допускающее с помощью функции **ISNULL(check\_expression, constant)**, где **constant** — ненулевое значение, заменяющее любой результат **NULL**.

## Принудительное обеспечение целостности данных

Планирование и создание таблиц требует указания допустимых значений для столбцов и определения способов принудительного обеспечения целостности данных в них. SQL Server предоставляет следующие механизмы для принудительного обеспечения целостности данных в столбце.

### Ограничения *PRIMARY KEY*

Обычно в таблице есть столбец или комбинация столбцов, содержащих значения, уникально определяющие каждую строку таблицы. Этот столбец, или столбцы, называются первичным ключом (ПК) таблицы и обеспечивает целостность сущности таблицы. Можно создать первичный ключ, задав ограничение *PRIMARY KEY* при создании или изменении таблицы.

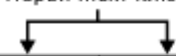
В таблице может быть только одно ограничение *PRIMARY KEY*, и столбец с данным ограничением не может содержать значения *NULL*. Из-за того, что ограничения *PRIMARY KEY* гарантируют уникальность данных, они часто определяются в столбце идентификаторов.

При задании для таблицы ограничения *PRIMARY KEY* компонент Database Engine гарантирует уникальность данных созданием уникального индекса для столбцов первичных ключей. Этот индекс также обеспечивает быстрый доступ к данным при использовании первичного ключа в запросах. Таким образом, выбранные первичные ключи должны соответствовать правилам [создания уникальных индексов](#).

Если ограничение *PRIMARY KEY* задано более чем для одного столбца, то значения могут дублироваться в пределах одного столбца, но каждая комбинация значений всех столбцов в определении ограничения *PRIMARY KEY* должна быть уникальной.

Как показано на следующем рисунке, столбцы **ProductID** и **VendorID** таблицы **Purchasing.ProductVendor** формируют составное ограничение *PRIMARY KEY* для данной таблицы. Это гарантирует уникальность комбинации значений столбцов **ProductID** и **VendorID**.

Первичный ключ



| ProductID | VendorID | AverageLeadTime | StandardPrice | LastReceiptCost |
|-----------|----------|-----------------|---------------|-----------------|
| 1         | 1        | 17              | 47.8700       | 50.2635         |
| 2         | 104      | 19              | 39.9200       | 41.9160         |
| 7         | 4        | 17              | 54.3100       | 57.0255         |
| 609       | 7        | 17              | 25.7700       | 27.0585         |
| 609       | 100      | 19              | 28.1700       | 29.5785         |

Таблица ProductVendor

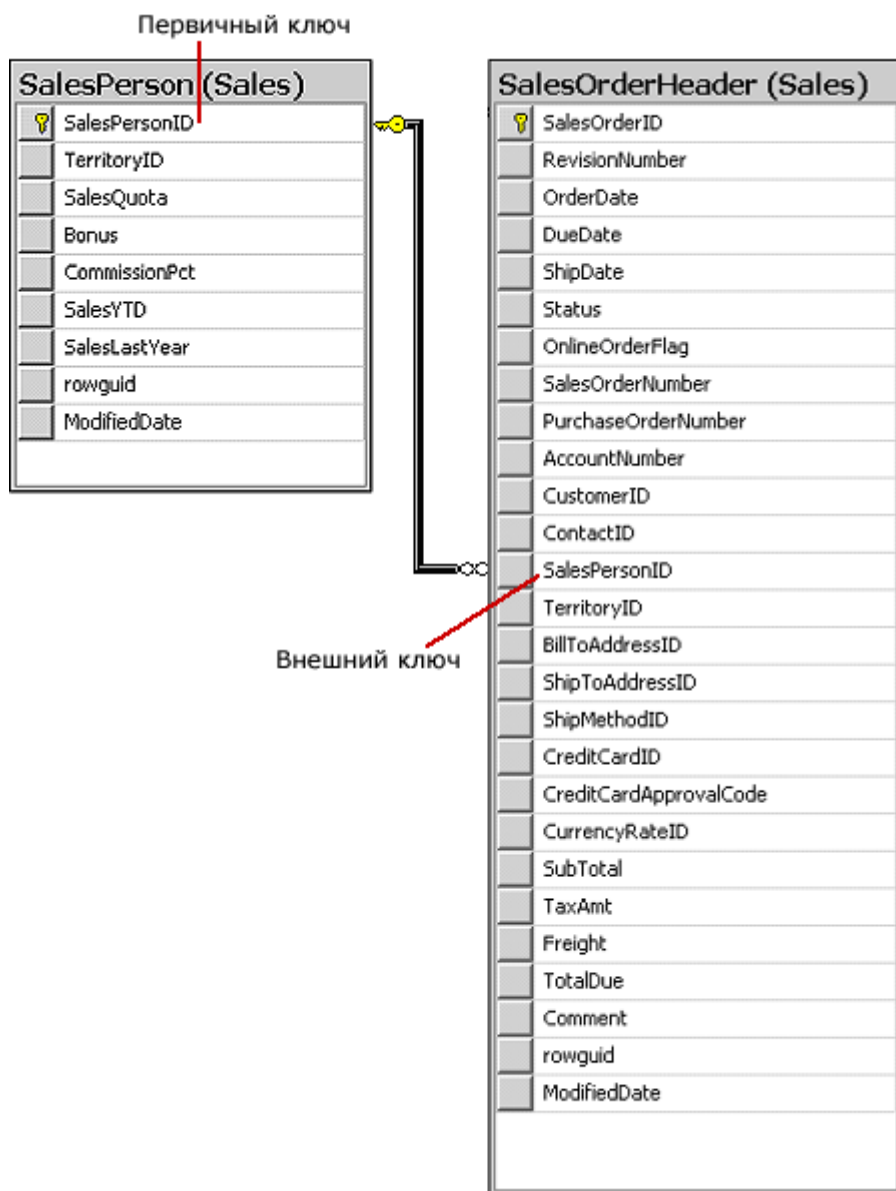
При работе с соединениями ограничения *PRIMARY KEY* связывают одну таблицу с другой. Например, для определения того, какие поставщики какие продукты поставляют, можно использовать тройное соединение таблиц **Purchasing.Vendor**, **Production.Product** и **Purchasing.ProductVendor**. Из-за того, что **ProductVendor** содержит одновременно столбцы **ProductID** и **VendorID**, доступ к таблице **Product** и к таблице **Vendor** может быть осуществлен благодаря их связи с таблицей **ProductVendor**.

### **Ограничения *FOREIGN KEY***

Внешний ключ (FK) — это столбец или сочетание столбцов, которое применяется для принудительного установления связи между данными в двух таблицах. Внешний ключ можно создать, определив ограничение FOREIGN KEY при создании или изменении таблицы.

Если один или несколько столбцов, в которых находится первичный ключ для одной таблицы, упоминается в одном или нескольких столбцах другой таблицы, то в ссылке внешнего ключа создается связь между двумя таблицами. Этот столбец становится внешним ключом во второй таблице.

Например, в таблице **Sales.SalesOrderHeader** в базе данных База данных AdventureWorks2008R2 есть связь с таблицей **Sales.SalesPerson**, так как существует логическая связь между заказами на продажу и менеджерами по продажам. Столбец **SalesPersonID** в таблице **Sales.SalesOrderHeader** соответствует столбцу первичного ключа в таблице **SalesPerson**. Столбец **SalesPersonID** в таблице **Sales.SalesOrderHeader** является внешним ключом для таблицы **SalesPerson**.



Ограничение FOREIGN KEY не обязательно должно быть связано с ограничением PRIMARY KEY в другой таблице. Кроме того, с помощью этого ограничения могут указываться столбцы ограничения UNIQUE в другой таблице. Ограничение FOREIGN KEY может содержать значения NULL. Впрочем, если любой из столбцов сложного ограничения FOREIGN KEY содержит значения NULL, то при проверке будут пропущены все значения, составляющие ограничение FOREIGN KEY. Чтобы проверялись все значения сложного ограничения FOREIGN KEY, укажите для всех участвующих столбцов параметр NOT NULL.

#### Примечание

Ограничение FOREIGN KEY может ссылаться на столбцы в таблицах этой же базы данных или столбцы этой же самой таблицы. Такие таблицы называются ссылающимися на себя. Возьмем, к примеру, таблицу работников, состоящую из трех столбцов: **employee\_number**, **employee\_name** и **manager\_employee\_number**. Поскольку менеджер также является работником, существует связь внешнего ключа в столбце **manager\_employee\_number** со столбцом **employee\_number**.

## Ссылочная целостность

Хотя главная задача ограничения FOREIGN KEY — управление данными, которые могут быть сохранены в таблице внешнего ключа, это ограничение также контролирует изменение данных в таблице первичного ключа. Например, при удалении строки для менеджера по продажам из таблицы **Sales.SalesPerson**, идентификатор которого используется в заказах на продажу в таблице **Sales.SalesOrderHeader**, ссылочная целостность двух таблиц будет нарушена. Заказы на продажу удаленного менеджера в таблице **SalesOrderHeader** станут недействительными без связи с данными в таблице **SalesPerson**.

Ограничение FOREIGN KEY предотвращает возникновение этой ситуации. Ограничение обеспечивает целостность ссылок следующим образом: оно запрещает изменение данных в таблице первичного ключа, если такие изменения сделают недопустимой ссылку в таблице внешнего ключа. Если при попытке удалить строку в таблице первичного ключа или изменить значение этого ключа окажется, что удаленному или измененному значению первичного ключа соответствует значение в ограничении FOREIGN KEY в другой таблице, то действие выполнено не будет. Для успешного изменения или удаления строки с ограничением FOREIGN KEY необходимо сначала удалить данные внешнего ключа в соответствующей таблице либо изменить данные в таблице внешнего ключа, которые связывают внешний ключ с данными другого первичного ключа.

## Индексирование ограничений FOREIGN KEY

Необходимость создать индекс внешних ключей часто возникает по следующим причинам:

- С помощью ограничений FOREIGN KEY в связанных таблицах проверяются изменения ограничений PRIMARY KEY.
- Столбцы внешних ключей часто используются для объединения критериев при соединении данных из связанных таблиц в запросах. Это делается путем сопоставления столбца или столбцов в ограничении FOREIGN KEY в одной таблице с одним или несколькими столбцами первичного или уникального ключей в другой таблице. Индекс позволяет компоненту Database Engine быстро находить связанные данные в таблице внешних ключей. Впрочем, создание индекса не является обязательным. Данные из двух связанных таблиц можно объединять, даже если в таблицах нет соответствующих ограничений PRIMARY KEY или FOREIGN KEY, однако связь внешних ключей между двумя таблицами означает, что эти две таблицы оптимизированы для соединения в запросе, где ключи используются в качестве критериев. Дополнительные сведения о применении ограничений FOREIGN KEY при соединении см. в разделах [Основные принципы соединения](#) и [Типы запросов и индексы](#).

## Количество ограничений FOREIGN KEY в таблице

В SQL Server не предусмотрено ни максимальное количество ограничений FOREIGN KEY в одной таблице (со ссылками на другие таблицы), ни максимальное число ограничений FOREIGN KEY из других таблиц, ссылающихся на одну таблицу. Тем не менее фактическое максимальное число ограничений FOREIGN KEY ограничено конфигурацией оборудования и структуры базы данных и приложения. Рекомендуется, чтобы таблица содержала не более 253 ограничений FOREIGN KEY, а также чтобы на



нее ссылалось не более 253 ограничений FOREIGN KEY. При разработке базы данных и приложений следует учитывать стоимость принудительных ограничений FOREIGN KEY.

### Каскадные ограничения ссылочной целостности

С помощью каскадных ограничений ссылочной целостности можно определять действия, которые SQL Server будет предпринимать, когда пользователь попытается удалить или обновить ключ, на который указывают еще существующие внешние ключи.

Предложения REFERENCES инструкций [CREATE TABLE](#) и [ALTER TABLE](#) поддерживают предложения ON DELETE и ON UPDATE. Каскадные действия могут также быть определены с помощью диалогового окна [Связи внешнего ключа](#):

- [ ON DELETE { NO ACTION | CASCADE | SET NULL | SET DEFAULT } ]
- [ ON UPDATE { NO ACTION | CASCADE | SET NULL | SET DEFAULT } ]

По умолчанию подразумевается действие NO ACTION, если предложения ON DELETE и ON UPDATE не указаны.

#### ON DELETE NO ACTION

Указывает, что при попытке удалить строку с ключом, на которую ссылаются внешние ключи в строках других таблиц, нужно сообщить об ошибке, а для инструкции DELETE выполнить откат.

#### ON UPDATE NO ACTION

Указывает, что при попытке обновить ключевое значение, на которое ссылаются внешние ключи в строках других таблиц, нужно сообщить об ошибке, а для инструкции UPDATE выполнить откат.

Действия CASCADE, SET NULL и SET DEFAULT позволяют удалять и обновлять значения ключей, влияющие на таблицы, в которых определены связи внешних ключей, приводящие к таблице, в которую вносятся изменения. Если каскадные ссылочные действия были также определены для целевых таблиц, то и там указанные каскадные действия будут применены при обновлении или удалении соответствующих строк. Значение CASCADE не может быть указано для внешних и первичных ключей в столбцах типа timestamp.

#### ON DELETE CASCADE

Указывает, что при попытке удалить строку с ключом, на которую ссылаются внешние ключи в строках других таблиц, все строки, содержащие эти внешние ключи, также должны быть удалены.

#### ON UPDATE CASCADE

Указывает, что при попытке обновить ключевое значение, на которое ссылаются внешние ключи в строках других таблиц, все значения, составляющие этот внешний ключ, также должны быть обновлены до нового значения ключа.

#### Примечание

Значение CASCADE не может быть указано, если столбец типа timestamp является частью внешнего или ссылочного ключа.

#### ON DELETE SET NULL

Указывает, что при попытке удалить строку с ключом, на которую ссылаются внешние ключи в строках других таблиц, все значения, составляющие эти внешние ключи, должны быть изменены на NULL. Чтобы выполнялось это ограничение, все столбцы внешних ключей целевой таблицы должны допускать значение NULL.

#### ON UPDATE SET NULL



Указывает, что при попытке обновить ключевое значение, на которое ссылаются внешние ключи в строках других таблиц, все значения, составляющие эти внешние ключи, должны быть изменены на NULL. Чтобы выполнялось это ограничение, все столбцы внешних ключей целевой таблицы должны допускать значение NULL.

#### ON DELETE SET DEFAULT

Указывает, что при попытке удалить строку с ключом, на которую ссылаются внешние ключи в строках других таблиц, все значения, составляющие эти внешние ключи, должны быть изменены на значения по умолчанию. Чтобы выполнялось это ограничение, для всех столбцов внешних ключей целевой таблицы должно быть определено значение по умолчанию. Если столбец допускает значение NULL и множество значений по умолчанию не задано явно, NULL становится неявным значением по умолчанию для данного столбца. Все задаваемые предложением ON DELETE SET DEFAULT значения, отличные от NULL, должны иметь соответствующие значения в основной таблице, чтобы сохранить целостность ограничения внешнего ключа.

#### ON UPDATE SET DEFAULT

Указывает, что при попытке обновить ключевое значение, на которое ссылаются внешние ключи в строках других таблиц, все значения, составляющие эти внешние ключи, должны быть изменены на значение по умолчанию. Чтобы выполнялось это ограничение, для всех столбцов внешних ключей целевой таблицы должно быть определено значение по умолчанию. Если столбец допускает значение NULL и множество значений по умолчанию не задано явно, NULL становится неявным значением по умолчанию для данного столбца. Все задаваемые предложением ON UPDATE SET DEFAULT значения, отличные от NULL, должны иметь соответствующие значения в основной таблице, чтобы сохранить целостность ограничения внешнего ключа.

Рассмотрим

ограничение **FK\_ProductVendor\_Vendor\_VendorID** таблицы **Purchasing.ProductVendor** базы данных База данных AdventureWorks2008R2. Это ограничение устанавливает связь внешнего ключа из столбца **VendorID** таблицы **ProductVendor** в первичный ключ **VendorID** таблицы **Purchasing.Vendor**. Если для этого ограничения указано действие ON DELETE CASCADE, удаление строки в таблице **Vendor**, где столбец **VendorID** имеет значение 100, приведет к удалению трех строк в таблице **ProductVendor**, где столбец **VendorID** также имеет значение 100. Если для этого ограничения указано действие ON UPDATE CASCADE, обновление значения столбца **VendorID** в таблице **Vendor** со 100 до 155 приведет к обновлению значений столбца **VendorID** в трех строках таблицы **ProductVendor**, где столбец **VendorID** также имеет значение 100.

Действие ON DELETE CASCADE не может быть указано в таблице, для которой определен триггер INSTEAD OF DELETE. В таблицах, для которых определены триггеры INSTEAD OF UPDATE, не могут быть указаны следующие действия: ON DELETE SET NULL, ON DELETE SET DEFAULT, ON UPDATE CASCADE, ON UPDATE SET NULL и ON UPDATE SET DEFAULT.

### **Множественные каскадные действия**

Отдельные инструкции DELETE или UPDATE могут начать серию каскадных ссылочных действий. Например, база данных содержит три таблицы: **TableA**, **TableB** и **TableC**. Внешний ключ таблицы **TableB** определен с действием ON DELETE CASCADE и связан с первичным ключом таблицы **TableA**. Внешний ключ таблицы **TableC** определен с

действием ON DELETE CASCADE и связан с первичным ключом таблицы **TableB**. Если инструкция DELETE удаляет строки в таблице **TableA**, эта же операция приведет к удалению всех строк в таблице **TableB**, в которых внешние ключи совпадают с удаленными первичными ключами таблицы **TableA**, а затем — к удалению всех строк в таблице **TableC**, в которых внешние ключи совпадают с удаленными первичными ключами таблицы **TableB**.

Последовательности каскадных ссылочных действий, запускаемые отдельными инструкциями DELETE или UPDATE, должны образовывать дерево без циклических ссылок. Никакая таблица не должна появляться больше одного раза в списке всех каскадных ссылочных действий, вызванных инструкциями DELETE или UPDATE. Кроме того, в дереве каскадных ссылочных действий к любой из задействованных таблиц должен быть только один путь. Любая ветвь в дереве прерывается, как только встречается таблица, для которой указано действие NO ACTION или вообще не указано действие.

### ***Триггеры и каскадные ссылочные действия***

Каскадные ссылочные действия запускают триггеры AFTER UPDATE или AFTER DELETE следующим образом:

Все каскадные ссылочные действия, прямо вызванные исходными инструкциями DELETE или UPDATE, выполняются первыми.

Если есть какие-либо триггеры AFTER, определенные для измененных таблиц, эти триггеры запускаются после выполнения всех каскадных действий. Эти триггеры запускаются в порядке, обратном каскадным действиям. Если для одной таблицы определены несколько триггеров, они запускаются в случайном порядке, если только не указаны выделенные первый и последний триггеры таблицы. Этот порядок определяется процедурой [sp\\_settriggerorder](#).

Если последовательности каскадных действий происходят из таблицы, которая была непосредственной целью действий DELETE или UPDATE, порядок запуска триггеров этими последовательностями действий не определен. Однако одна последовательность действий всегда запускает все свои триггеры до того, как это начнет делать следующая.

Триггер AFTER таблицы, являвшейся непосредственной целью действий DELETE или UPDATE, запускается вне зависимости от того, были ли изменены хоть какие-нибудь строки. В этом случае ни на какие другие таблицы каскадирование не влияет.

Если один из предыдущих триггеров выполняет операции DELETE или UPDATE над другими таблицами, эти операции могут вызывать собственные последовательности каскадных действий. Эти вторичные последовательности действий обрабатываются для каждой операции DELETE или UPDATE после выполнения всех триггеров первичных последовательностей действий. Этот процесс может рекурсивно повторяться для последующих операций DELETE или UPDATE.

Выполнение операций CREATE, ALTER, DELETE или других операций языка DDL внутри триггеров может привести к запуску триггеров DDL. Это может привести к дальнейшим операциям DELETE или UPDATE, которые начнут дополнительные последовательности

каскадных действий и запустят свои триггеры.

Если в любой конкретной последовательности каскадных ссылочных действий произойдет ошибка, в этой последовательности не будут запущены никакие триггеры AFTER, а для операций DELETE или UPDATE, создаваемых этой последовательностью, будет выполнен откат.

У таблицы, для которой определен триггер INSTEAD OF, может также быть предложение REFERENCES, указывающее конкретное каскадное действие. Однако триггер AFTER целевой таблицы каскадного действия может выполнить инструкцию INSERT, UPDATE или DELETE для другой таблицы или представления, которое запустит триггер INSTEAD OF для этого объекта.

### **Сведения из каталога каскадных ссылочных ограничений**

Запрос к представлению каталога **sys.foreign\_keys** вернет следующие значения, описывающие каскадные ссылочные ограничения, указанные для внешних ключей.

| Значение | Описание    |
|----------|-------------|
| 0        | NO ACTION   |
| 1        | CASCADE     |
| 2        | SET NULL    |
| 3        | SET DEFAULT |

Столбцы **UPDATE\_RULE** и **DELETE\_RULE**, возвращаемые процедурами **sp\_fkeys** и **sp\_foreignkeys**, содержат 0, если указаны действия CASCADE, SET NULL или SET DEFAULT, и 1, если указано действие NO ACTION или используется значение по умолчанию.

Если хранимой процедуре **sp\_help** передать внешний ключ как объект, то результирующий набор будет содержать следующие столбцы.

| Имя столбца          | Тип данных  | Описание                                                                                                                 |
|----------------------|-------------|--------------------------------------------------------------------------------------------------------------------------|
| <b>delete_action</b> | nvarchar(9) | Указывает, какое действие используется для удаления: CASCADE, SET NULL, SET DEFAULT, NO ACTION или N/A (не применимо).   |
| <b>update_action</b> | nvarchar(9) | Указывает, какое действие используется для обновления: CASCADE, SET NULL, SET DEFAULT, NO ACTION или N/A (не применимо). |

## Ограничения **UNIQUE**

Ограничения **UNIQUE** можно использовать, чтобы убедиться, что в отдельные столбцы, не включенные в первичный ключ, не вводятся повторяющиеся значения. Хотя уникальность значений ограничения **UNIQUE** и **PRIMARY KEY** гарантируют в равной степени, в случае, когда необходимо обеспечить уникальность в столбце или комбинации столбцов, которые не являются первичными ключевыми, вместо ограничения **PRIMARY KEY** следует использовать ограничение **UNIQUE**.

Для таблицы можно задать несколько ограничений **UNIQUE**, и только одно ограничение **PRIMARY KEY**.

Также, в отличие от **PRIMARY KEY**, ограничения **UNIQUE** допускают значение **NULL**. Однако, как и всякое другое значение столбца с ограничением **UNIQUE**, **NULL** может встречаться только один раз.

На ограничение **UNIQUE** могут ссылаться ограничения **FOREIGN KEY**.

## Ограничения **CHECK**

Проверочные ограничения обеспечивают доменную целостность, ограничивая значения, которые может принимать столбец. Аналогично ограничениям внешнего ключа, они управляют значениями, которые присваиваются столбцу. Однако они по-разному определяют допустимые значения: ограничения внешнего ключа получают список допустимых значений из другой таблицы, а проверочные ограничения определяют допустимые значения по логическому выражению, которое не основано на данных в другом столбце. Например, чтобы ограничить интервал значений столбца **salary**, можно создать проверочное ограничение, позволяющее столбцу принимать значения только в интервале от 15 до 100 тыс. долларов. Это ограничение исключает возможность устанавливать размер зарплаты, отличный от обычного.

Проверочное ограничение можно создать с любым логическим выражением, возвращающим значение **TRUE** или **FALSE** на основе логических операторов. Для предыдущего примера логическое выражение будет выглядеть следующим образом: `salary >= 15000 AND salary <= 100000`.

К одному столбцу можно применять несколько проверочных ограничений. Кроме того, можно применять одно проверочное ограничение к нескольким столбцам. Для этого ограничение нужно создать на уровне таблицы. Например, с помощью проверочного ограничения на несколько столбцов можно подтвердить то, что любая строка со значением **USA** в столбце **country/region** может принимать двухсимвольное значение в столбце **state**. Это позволяет выполнить проверку сразу нескольких условий из одного выражения.

### Внимание!

Ограничения, которые включают явное или неявное преобразование данных, могут вызывать ошибки в операциях такого рода. Например, ограничения, заданные для таблиц, которые являются исходными при переключении секций, могут приводить к ошибкам при использовании оператора **ALTER TABLE...SWITCH**. Следует избегать

преобразования типов данных в определениях ограничений.

### Ограничения проверочных ограничений

Проверочные ограничения отклоняют значения, вычисляемые в FALSE. Поскольку значения NULL вычисляются как UNKNOWN, то их наличие в выражениях может переопределить ограничение. Например, предположим, что на столбец **MyColumn** типа `int` установлено следующее ограничение: **MyColumn** может содержать только значение 10 (**MyColumn=10**). При вставке значения NULL в столбец **MyColumn** компонент Database Engine вставит значение NULL и не возвратит ошибку.

Проверочное ограничение возвращает TRUE, если для проверяемого условия в любой строке таблицы отсутствует значение FALSE. Если в только что созданной таблице отсутствуют строки, то любое проверочное ограничение на эту таблицу считается допустимым. В результате могут возвращаться неожиданные результаты, как в следующем примере.

```
CREATE TABLE CheckTbl (col1 int, col2 int);
GO
CREATE FUNCTION CheckFnctn()
RETURNS int
AS
BEGIN
    DECLARE @retval int
    SELECT @retval = COUNT(*) FROM CheckTbl
    RETURN @retval
END;
GO
ALTER TABLE CheckTbl
ADD CONSTRAINT chkRowCount CHECK (dbo.CheckFnctn() >= 1 );
GO
```

Ограничение CHECK показывает, что в таблице CheckTbl должна быть хотя бы одна строка. Однако поскольку в таблице нет ни одной строки, над которой можно было бы произвести проверку ограничения, инструкция ALTER TABLE завершается успешно.

Проверочные ограничения не проверяются во время выполнения инструкций DELETE. Таким образом, выполнение инструкций DELETE над таблицами с некоторыми типами проверочных ограничений может приводить к неожиданным результатам. Например, предположим, что следующие выражения выполняются над таблицей CheckTbl.

```
INSERT INTO CheckTbl VALUES (10, 10)
GO
DELETE CheckTbl WHERE col1 = 10;
```

Инструкция DELETE выполняется успешно даже притом, что ограничением CHECK задано, что в таблице CheckTbl должна хотя бы 1 строка.

### Определения DEFAULT

Каждый столбец записи должен содержать значение, даже если это значение равно NULL. Возможны случаи, когда необходимо поместить данные в таблицу, однако

значение какого-либо столбца неизвестно, или же это значение не существует. Если столбец допускает значения NULL, то можно поместить в него значение NULL.

Поскольку иногда помещать в столбцы значения NULL нежелательно, лучшим решением может оказаться определение для столбца значения по умолчанию (DEFAULT).

Например для числовых столбцов обычно в качестве значения по умолчанию указывается ноль, а для символьных — значение «Н/Д».

При загрузке строки в таблицу, имеющую для столбца определение DEFAULT, Database Engine неявно выполняет в этот столбец вставку значения по умолчанию, если оно явно не указано.

#### Примечание

Можно также явно указать компоненту Database Engine, чтобы он использовал значение столбца по умолчанию при вставке данных, воспользовавшись предложением DEFAULT VALUES инструкции INSERT.

Если столбец не допускает значений NULL и определение DEFAULT для него отсутствует, необходимо явно указать значение столбца, в противном случае компонент Database Engine возвратит сообщение об ошибке, указывающее, что значения NULL в столбце не допускаются.

Вставляемое в столбец значение определяется совместно определением DEFAULT и возможностью помещать в столбец значения NULL. Сводка результатов при различных комбинациях этих параметров приведена в следующей таблице.

| Определение столбца     | Запись отсутствует, нет определения DEFAULT | Запись отсутствует, есть определение DEFAULT | Ввод значения NULL |
|-------------------------|---------------------------------------------|----------------------------------------------|--------------------|
| Значения NULL допустимы | NULL                                        | Значение по умолчанию                        | NULL               |
| Значения NULL запрещены | Ошибка                                      | Значение по умолчанию                        | Ошибка             |

### Разрешение значений NULL

Допустимость значения NULL в столбце определяет, могут ли строки таблицы содержать значения NULL для этого столбца. Значение NULL отличается от нуля (0), пробела или символьной строки нулевой длины, например "". Значение NULL обозначает, что поле не было заполнено. Присутствие NULL, как правило, подразумевает, что значение неизвестно или неопределенно. Например, значение NULL в столбце **SellEndDate** таблицы **Production.Product** базы данных База данных AdventureWorks2008R2 не означает, что это изделие не имеет даты окончания продаж, а говорит о том, что дата неизвестна или не была задана.

#### Примечание

Рекомендуется избегать разрешать использование значений NULL, поскольку они

усложняют выполнение запросов и процедур обновления. Кроме того, для столбцов со значением NULL не могут быть использованы некоторые параметры, например ограничения PRIMARY KEY.

Когда в таблицу вставляется строка без указания значения для столбца, допускающего значения NULL, компонент Database Engine присваивает ему значение NULL, если отсутствует определение DEFAULT или объект по умолчанию. Столбец, определенный с помощью ключевого слова NULL, также поддерживает явно заданное пользователем значение NULL, независимо от типа данных и наличия соответствующего значения по умолчанию. Значение NULL не следует заключать в кавычки, так как в этом случае оно будет рассматриваться как символьная строка «NULL».

Запрет значений NULL в столбце обеспечивает целостность данных, гарантируя, что столбец в строке будет всегда содержать данные. Если значения NULL запрещены, то пользователь, заполняя таблицу данными, должен ввести значения в столбцы, иначе строка таблицы не будет записана в базу данных.

#### Примечание

Столбцы, определенные с помощью ограничения PRIMARY KEY или свойства IDENTITY, не допускают значений NULL.

## Создание и изменение таблиц

### Основы создания и изменение таблиц

После построения базы данных можно создать таблицы, в которых будут храниться данные. Данные хранятся главным образом в постоянных таблицах, но можно создавать и временные. Таблицы хранятся в базе данных, пока их не удалят, и доступны любому пользователю, имеющему соответствующие разрешения.

#### Свойства таблицы

Каждая таблица может содержать до 1024 столбцов. Имена таблиц и столбцов должны соответствовать правилам для идентификаторов; они должны быть уникальными в пределах таблицы, другими словами, можно использовать одинаковые имена столбцов в разных таблицах одной базы данных. Дополнительные сведения см. в разделах [Идентификаторы](#) и [Использование идентификаторов в качестве имен объектов](#). Для каждого столбца необходимо определить тип данных. Дополнительные сведения см. в разделе [Присвоение типа данных столбцу](#).

Имена столбцов должны быть уникальными для каждой схемы в базе данных, поэтому можно создать несколько таблиц с одинаковыми именами, указав для каждой свою схему. Например, можно создать две таблицы **employees** и назначить первой схему **Jonah**, а второй схему **Sally**. Чтобы обратиться к нужной таблице **employees**, необходимо указать ее схему.

Создание таблицы

## Инструкция CREATE TABLE (Transact-SQL)

### Как создавать таблицы (визуальные инструменты для баз данных)

#### **Временные таблицы**

Временные таблицы отличаются от постоянных только тем, что хранятся в базе данных **tempdb** и автоматически удаляются, когда необходимость в них отпадает.

Существует два вида временных таблиц: локальные и глобальные. Они отличаются друг от друга именами, видимостью и доступностью. Имена локальных временных таблиц начинаются с одного символа (#); они видны только текущему соединению пользователя и удаляются, когда пользователь отключается от экземпляра SQL Server. Имена глобальных таблиц начинаются с двух символов номера (##); они видны любому пользователю и удаляются, когда все пользователи, которые на них ссылаются, отключаются от экземпляра SQL Server.

Например, если создать таблицу **employees**, она будет доступна любому пользователю, которому предоставлены разрешения на ее использование до тех пор, пока не будет удалена. Если во время сеанса базы данных создается локальная временная таблица **#employees**, с ней сможет работать только данный сеанс. Таблица будет удалена при завершении сеанса. Если создать глобальную временную таблицу **##employees**, с ней сможет работать любой пользователь базы данных. Если другие пользователи не будут работать с этой таблицей, она будет удалена после отключения от нее. Если другой пользователь обратится к созданной таблице, SQL Server удалит ее, когда произойдет отключение и другие сеансы перестанут активно к ней обращаться.

Если временная таблица создана с именованным ограничением и в области определяемой пользователем транзакции, то только один пользователь может одновременно выполнить инструкцию, создающую временную таблицу. Например, если хранимая процедура создает временную таблицу с ограничением именованного первичного ключа, то хранимая процедура не может быть одновременно выполнена несколькими пользователями.

В большинстве случаев временные таблицы можно заменить переменными типа **table**. Дополнительные сведения об использовании переменных типа **table** см. [table \(Transact-SQL\)](#).

#### **Изменение таблиц**

Большинство параметров, которые были указаны при первоначальном создании таблицы, можно изменить. Можно выполнить следующие действия.

- Можно добавить, изменить или удалить столбцы. У столбца можно изменить имя, длину, тип данных, точность, масштаб и возможность принимать значения NULL, хотя существуют некоторые ограничения. Дополнительные сведения см. в разделе [Изменение свойств столбцов](#).
- В секционированной таблице можно перераспределить, добавить или удалить отдельные секции. Дополнительные сведения см. в разделах [Изменение секционированных таблиц и индексов](#) и [Эффективная передача данных с использованием переключения секций](#).



- Ограничения PRIMARY KEY и FOREIGN KEY можно добавить или удалить.
- Ограничения UNIQUE и CHECK, определение DEFAULT и объекты можно добавить или удалить.
- Столбец идентификаторов можно добавить или удалить, используя свойство IDENTITY или ROWGUIDCOL. Свойство ROWGUIDCOL можно добавить или удалить из существующего столбца, хотя таблица может содержать только один столбец со свойством ROWGUIDCOL.
- Таблицы и выбранные столбцы таблицы можно зарегистрировать для полнотекстовой индексации.

Дополнительные сведения об изменении таблиц см. [ALTER TABLE \(Transact-SQL\)](#).

Имя или схему таблицы также можно изменить. В этом случае необходимо изменить имя таблицы во всех триггерах, хранимых процедурах, сценариях Transact-SQL и другом программном коде, где используется старое имя или схема таблицы.

## Изменение свойств столбцов

Каждый столбец в таблице обладает набором таких свойств, как имя, тип данных, возможность наличия значений NULL, длина данных. Полный набор свойств столбца образует определение столбца в таблице.

### *Тип данных столбца*

Тип данных существующего столбца может быть изменен при условии, что существующие данные в столбце могут быть неявно преобразованы в новый тип данных. Дополнительные сведения см. в разделе [ALTER TABLE \(Transact-SQL\)](#).

### *Длина данных столбца*

При выборе типа данных автоматически определяется длина столбца. Увеличить или уменьшить длину можно только для столбцов типов данных binary, char, nchar, varbinary, varchar и nvarchar. Для столбцов других типов длина вычисляется на основе типа данных и не может быть изменена. Если новая задаваемая длина меньше исходной длины столбца, все значения столбца, длина которых превышает новую длину, усекаются без какого-либо предупреждения. Нельзя изменить длину столбца, определенного с ограничением PRIMARY KEY или FOREIGN KEY.

### *Точность столбца*

Точность числового столбца определяется максимальным количеством цифр, используемых в выбранном типе данных. Точность нечислового столбца обычно определяется или максимальной длиной, или определенной длиной столбца.

Для всех типов данных, кроме decimal и numeric, точность определяется автоматически. Если необходимо переопределить максимальное количество цифр, используемых в значениях столбцов типов данных decimal и numeric, можно изменить точность этих столбцов. Компонент SQL Server Database Engine предотвращает изменение точности столбца, тип данных которого отличается от указанных типов данных.

## Масштаб столбца

Масштаб столбца numeric или decimal определяется максимальным количеством цифр справа от десятичной запятой. При выборе типа данных масштаб столбца по умолчанию устанавливается в 0. Для столбцов с приближительными числами с плавающей запятой масштаб не определен, так как количество цифр справа от десятичной запятой не фиксировано. Можно изменить масштаб столбца типа numeric или decimal, если необходимо переопределить количество цифр, стоящих справа от десятичной запятой.

## Возможность наличия в столбце значений NULL

Столбец может быть определен как допускающий значения NULL или не допускающий их. По умолчанию в столбце допускается наличие значений NULL. Существующий столбец может быть изменен, чтобы в нем значения NULL не допускались, если только в столбце еще нет таких значений и на столбце не создано никакого индекса. Для запрета возможности наличия значений NULL в существующем столбце, который их содержит, необходимо выполнить следующие шаги.

1. Добавьте новый столбец с определением DEFAULT, которое вместо значения NULL вставляет допустимое значение.
2. Скопируйте данные старого, уже существующего столбца в новый столбец.
3. Удалите старый столбец.

Существующий столбец, в котором не допускается наличие значений NULL, может быть изменен, чтобы в нем допускались значения NULL, если только на столбце не определено ограничение PRIMARY KEY.

## Разреженные столбцы и наборы столбцов

Разреженные столбцы — это столбцы, в большинстве строк которых содержатся значения NULL. Разреженные столбцы следует использовать в том случае, если от 20 до 40 процентов от всех значений столбца составляют значения NULL. Разреженные столбцы — это обычные столбцы, имеющие оптимизированное хранилище для значений NULL. Дополнительные сведения см. в разделе [Использование разреженных столбцов](#).

В таблицах, использующих разреженные столбцы, можно назначить набор столбцов, который будет возвращать все разреженные столбцы в таблице. Набор столбцов — это нетипизированное XML-представление, которое объединяет на выходе все разреженные столбцы таблицы в структурированном виде. Набор столбцов похож на вычисляемые столбцы тем, что набор столбцов физически не хранится в таблице. Дополнительные сведения см. в разделе [Использование наборов столбцов](#).

Задание свойств столбца

- [ALTER TABLE \(Transact-SQL\)](#)
- [Изменение столбцов \(визуальные инструменты для баз данных\)](#)

Просмотр свойств столбца

- [COLUMNPROPERTY \(Transact-SQL\)](#)
- [Свойства столбца таблицы \(среда SQL Server Management Studio\)](#)

Переименование столбца

- [sp\\_rename \(Transact-SQL\)](#)
- [Как переименовать столбцы \(визуальные инструменты для баз данных\)](#)

## Добавление и удаление столбцов

К существующим таблицам можно добавлять столбцы, указывая, разрешены ли для них значения NULL и применимы ли ограничения DEFAULT. При добавлении нового столбца в таблицу компонент Database Engine вставляет в этот столбец значение для каждой существующей в таблице строки данных. Поэтому при добавлении столбца в таблицу полезно добавить к этому столбцу определение DEFAULT. Если для нового столбца нет определения DEFAULT, необходимо указать, что в этом столбце допускаются значения NULL. Компонент Database Engine вставит в столбец значения NULL или вернет ошибку, если для нового столбца значения NULL запрещены.

Также можно удалять столбцы из существующих таблиц, кроме столбцов со следующими характеристиками:

- используемые в индексе;
- используемые в ограничениях CHECK, FOREIGN KEY, UNIQUE или PRIMARY KEY;
- связанные с определением DEFAULT или привязанные к объекту по умолчанию;
- привязанные к правилу;
- зарегистрированные для полнотекстовой поддержки;
- используемые в качестве полнотекстового ключа для таблицы.

Дополнительные сведения о добавлении и удалении столбцов из таблиц, опубликованных для репликации, см. в подразделах «Добавление столбцов» и «Удаление столбцов» раздела [Внесение изменений схем в базы данных публикаций](#).

Добавление или удаление столбца

[ALTER TABLE \(Transact-SQL\)](#)

[Как вставлять столбцы в таблицы \(визуальные инструменты для баз данных\)](#)

[Как удалять столбцы из таблицы \(визуальные инструменты для баз данных\)](#)

## Использование разреженных столбцов

Разреженные столбцы — это обычные столбцы, имеющие оптимизированное хранилище для значений NULL. Разреженные столбцы уменьшают пространство, необходимое для хранения значений NULL, однако увеличивается стоимость получения значений, отличных от NULL. Разреженные столбцы следует использовать только в том случае, если экономится не менее чем от 20 до 40 процентов места. Наборы столбцов и разреженные столбцы определяются с помощью инструкций [CREATE TABLE](#) и [ALTER TABLE](#).

Разреженные столбцы можно использовать совместно с наборами столбцов и фильтруемые индексами.

- Наборы столбцов  
Инструкции INSERT, UPDATE и DELETE могут ссылаться на разреженные столбцы

по именам. Также можно просматривать и работать со всеми разреженными столбцами таблицы, объединенными в один XML-столбец. Такой столбец называется набором столбцов. Дополнительные сведения о наборах столбцов см. в разделе [Использование наборов столбцов](#).

- **Фильтруемые индексы**

Поскольку разреженные столбцы имеют много строк со значениями NULL, они особенно подходят для фильтруемых индексов. Фильтруемый индекс на основе разреженного столбца может индексировать только те строки, значения которых заполнены. Таким образом создается более компактный и эффективный индекс. Дополнительные сведения см. в разделе [Рекомендации по проектированию отфильтрованных индексов](#).

Разреженные столбцы и фильтруемые индексы позволяют приложениям, таким как Windows SharePoint Services, эффективно хранить и получать доступ к большому числу определяемых пользователем свойств с помощью SQL Server.

### *Свойства разреженных столбцов*

Разреженные столбцы имеют следующие характеристики.

- Компонент SQL Server Database Engine использует ключевое слово SPARSE в определении столбца, чтобы оптимизировать хранение значений в этом столбце. Следовательно, если в любой строке таблицы этот столбец содержит значение NULL, место для хранения этих значений не требуется.
- Представления каталога таблицы, имеющей разреженные столбцы, идентичны представлениям обычной таблицы. Представление каталога **sys.columns** содержит по одной строке для каждого столбца в таблице, включая набор столбцов, если он был определен.
- Разреженные столбцы являются свойством уровня хранилища, а не логической таблицы. Поэтому инструкция SELECT...INTO не копирует свойство разреженного столбца в новую таблицу.
- Функция COLUMNS\_UPDATED возвращает значение типа varbinary, показывающее все столбцы, которые были обновлены в процессе DML-действия. Функция COLUMNS\_UPDATED возвращает следующие биты.
  - Если разреженный столбец был явно обновлен, соответствующий бит, представляющий этот столбец, и бит, представляющий набор столбцов, устанавливаются в 1.
  - Если набор столбцов был явно обновлен, бит, представляющий набор столбцов, и биты, представляющие все разреженные столбцы в таблице, устанавливаются в 1.
  - При операциях вставки всем битам присваивается значение 1.

Дополнительные сведения о наборах столбцов см. в разделе [Использование наборов столбцов](#).

Столбцы следующих типов данных не могут быть указаны как SPARSE.

|           |                         |
|-----------|-------------------------|
| geography | text                    |
| geometry  | timestamp               |
| image     | user-defined data types |
| ntext     |                         |

## Предполагаемая экономия места по типам данных

Для хранения значений, отличных от NULL, в разреженных столбцах требуется больше места, чем для хранения идентичных данных, но не отмеченных, как SPARSE. В следующих таблицах показано использование пространства для каждого типа данных. Столбец Процент значений NULL показывает, какой процент данных должен содержать значения NULL для достижения общей экономии пространства в 40 процентов.

Типы данных фиксированной длины

| Тип данных       | Неразрезанные байты | Разреженные байты | Процент значений NULL |
|------------------|---------------------|-------------------|-----------------------|
| bit              | 0.125               | 5                 | 98%                   |
| tinyint          | 1                   | 5                 | 86%                   |
| smallint         | 2                   | 6                 | 76%                   |
| int              | 4                   | 8                 | 64%                   |
| bigint           | 8                   | 12                | 52%                   |
| real             | 4                   | 8                 | 64%                   |
| float            | 8                   | 12                | 52%                   |
| smallmoney       | 4                   | 8                 | 64%                   |
| money            | 8                   | 12                | 52%                   |
| smalldatetime    | 4                   | 8                 | 64%                   |
| datetime         | 8                   | 12                | 52%                   |
| uniqueidentifier | 16                  | 20                | 43%                   |
| date             | 3                   | 7                 | 69%                   |

Типы данных с длиной, зависящей от точности

| Тип данных   | Неразрезанные байты | Разреженные байты | Процент значений NULL |
|--------------|---------------------|-------------------|-----------------------|
| datetime2(0) | 6                   | 10                | 57%                   |
| datetime2(7) | 8                   | 12                | 52%                   |

|                       |                                                           |    |     |
|-----------------------|-----------------------------------------------------------|----|-----|
| time(0)               | 3                                                         | 7  | 69% |
| time(7)               | 5                                                         | 9  | 60% |
| datetimeoffset(0)     | 8                                                         | 12 | 52% |
| datetimeoffset (7)    | 10                                                        | 14 | 49% |
| decimal/numeric(1,s)  | 5                                                         | 9  | 60% |
| decimal/numeric(38,s) | 17                                                        | 21 | 42% |
| vardecimal(p,s)       | В качестве консервативной оценки используйте тип decimal. |    |     |

Типы данных с длиной, зависящей от данных

| Тип данных           | Неразрезанные байты              | Разрезанные байты | Процент значений NULL |
|----------------------|----------------------------------|-------------------|-----------------------|
| sql_variant          | Зависит от базового типа данных. |                   |                       |
| varchar или char     | 2*                               | 4*                | 60%                   |
| nvarchar или nchar   | 2*                               | 4*+               | 60%                   |
| varbinary или binary | 2*                               | 4*                | 60%                   |
| xml                  | 2*                               | 4*                | 60%                   |
| hierarchyid          | 2*                               | 4*                | 60%                   |

\*Длина равна средней длине данных, содержащихся в типе, плюс 2 или 4 байта.

### **Издержки в памяти при выполнении операций обновления разреженных столбцов**

При проектировании таблиц с разреженными столбцами не забывайте о том, что при выполнении операции обновления строки требуется 2 дополнительных байта на каждый разреженный ненулевой столбец в таблице. Из-за этого операция обновления может непредвиденно завершиться ошибкой 576 в том случае, если общий размер строки, включая дополнительную память, превысит 8019 байт и будут отсутствовать столбцы, которые можно будет принудительно разместить вне строки.

Рассмотрим пример таблицы, которая содержит 600 разреженных столбцов типа bigint.

При наличии 571-го ненулевого столбца общий размер на диске составит  $571 * 12 = 6852$  байт. После добавления дополнительного размера строки и заголовка разреженного столбца размер увеличится до 6895 байт. Страница по-прежнему занимает около 1124 байт на диске. При этом создается впечатление, что дополнительные столбцы могут быть успешно обновлены. Но во время обновления появляются дополнительные издержки в памяти, которые составят  $2 * (\text{число разреженных ненулевых столбцов})$ . В данном примере размер строки на диске увеличится приблизительно до 8037 байт, включая дополнительные издержки  $2 * 571 = 1142$  байта. Это значение превышает максимально допустимый размер 8019 байт. Так как все столбцы содержат тип данных фиксированной длины, они не могут быть принудительно отправлены вне строки. В результате этой операция обновления завершится ошибкой 576.

### **Ограничения на использование разреженных столбцов**

Разреженные столбцы могут иметь любой тип данных SQL Server. Они работают так же, как и другие столбцы, но со следующими ограничениями.

- Разреженный столбец должен допускать значения NULL и не может иметь свойств ROWGUIDCOL или IDENTITY.
- Разреженный столбец не может иметь следующие типы данных: text, ntext, image, timestamp, определяемый пользователем тип данных, geometry или geography; также он не может иметь атрибут FILESTREAM.
- Разреженный столбец не может иметь значения по умолчанию.
- Разреженный столбец не может быть привязан к правилу.
- Хотя вычисляемый столбец и может содержать разреженный столбец, но сам вычисляемый столбец не может быть отмечен как SPARSE.
- Разреженный столбец не может быть частью кластеризованного индекса или индекса уникального первичного ключа. Однако материализованные и нематериализованные вычисляемые столбцы, определенные для разреженных столбцов, могут быть частью кластеризованного ключа.
- Разреженный столбец не может быть использован в качестве ключа секции для кластеризованного индекса или кучи. Однако разреженный столбец может быть использован в качестве ключа секции для некластеризованного индекса.
- Разреженный столбец не может быть частью определяемого пользователем табличного типа, используемого в переменных таблицы и в возвращающих табличное значение параметрах.
- Разреженные столбцы несовместимы со сжатием данных. Поэтому разреженные столбцы нельзя добавить в сжатые таблицы, а таблицы с разреженными столбцами нельзя сжать.
- Преобразование столбца из разреженного в неразреженный (или наоборот) требует изменения формата хранения столбца. Для внесения этого изменения компонент SQL Server Database Engine использует следующую процедуру.
  1. В таблицу добавляется новый столбец с новым размером хранения и форматом.
  2. Для каждой строки в таблице производится обновление и копирование значений, хранимых в старом столбце, в новый столбец.
  3. Из схемы таблицы удаляется старый столбец.
  4. Производится перестроение таблицы для освобождения места, используемого старым столбцом.

#### **Примечание**

Шаг 2 может завершиться неудачно, если размер данных в строке превышает максимально допустимый размер строки. Этот размер включает размер данных, хранимых в старом столбце, и обновленных данных, хранимых в новом столбце. Данное ограничение составляет 8 060 байт для таблиц, не содержащих разреженные столбцы, и 8 018 байт для таблиц, содержащих их. Данная ошибка может возникнуть, даже если все подходящие столбцы включают встроочные данные. Дополнительные сведения см. в разделе [Превышающие размер страницы данные строки, превышающие 8 КБ](#).

- При преобразовании неразреженного столбца в разреженный возрастет пространство, необходимое для хранения значений, отличных от NULL. Если строка имеет длину, близкую к максимальной, операция может завершиться неудачно.

### *Технологии SQL Server, поддерживающие разреженные столбцы*

В этом разделе описывается поддержка разреженных столбцов в следующих технологиях SQL Server.

- Репликация транзакций  
Репликация транзакций поддерживает разреженные столбцы, однако не поддерживает наборы столбцов, которые могут быть использованы с разреженными столбцами. Дополнительные сведения о наборах столбцов см. в разделе [Использование наборов столбцов](#).  
Репликация атрибута SPARSE определяется параметром схемы, задаваемым с помощью процедуры [sp\\_addarticle](#) либо с помощью диалогового окна Свойства статьи в среде Средства SQL Server Management Studio. Более ранние версии SQL Server не поддерживают разреженные столбцы. Если необходимо реплицировать данные в более раннюю версию, следует указать, что атрибут SPARSE не подлежит репликации.  
В опубликованные таблицы нельзя добавлять новые разреженные столбцы или изменять свойство SPARSE существующих столбцов. Если необходимо выполнить подобную операцию, следует удалить и повторно создать публикацию.
- Репликация слиянием  
Репликация слиянием не поддерживает разреженные столбцы и наборы столбцов.
- Отслеживание изменений  
Отслеживание изменений поддерживает разреженные столбцы и наборы столбцов. Если в таблице обновляется набор столбцов, система отслеживания изменений считает это обновлением целой строки. Более подробное отслеживание изменений для определения точного набора разреженных столбцов, который был изменен в ходе операции обновления набора столбцов, не осуществляется. Если разреженные столбцы обновляются явно с помощью инструкции DML, система отслеживания изменений обрабатывает их обычным образом и можно идентифицировать точный набор измененных столбцов.
- Система отслеживания измененных данных  
Система отслеживания измененных данных поддерживает разреженные столбцы, но не поддерживает наборы столбцов.
- При копировании таблицы свойство разреженности столбца не сохраняется.

### *Примеры*

В данном примере таблица документа содержит обычный набор со столбцами DocID и Title. Производственной группе необходимы столбцы ProductionSpecification и ProductionLocation для всех рабочих документов. Группе сбыта необходим столбец MarketingSurveyGroup для документов сбыта. Код из



этого примера создает таблицу, использующую разреженные столбцы, вставляет в таблицу две строки, затем выбирает из таблицы данные.

### Примечание

Эта таблица насчитывает лишь пять столбцов, что упрощает ее отображение и чтение. При установленном параметре ANSI\_NULL\_DFLT\_ON объявлять разреженные столбцы допускающими значения NULL необязательно.

```
USE AdventureWorks2008R2;
GO
```

```
CREATE TABLE dbo.DocumentStore
(DocID int PRIMARY KEY,
Title varchar(200) NOT NULL,
ProductionSpecification varchar(20) SPARSE NULL,
ProductionLocation smallint SPARSE NULL,
MarketingSurveyGroup varchar(20) SPARSE NULL );
GO
```

```
INSERT dbo.DocumentStore(DocID, Title, ProductionSpecification,
ProductionLocation)
VALUES (1, 'Tire Spec 1', 'AXZZ217', 27);
GO
```

```
INSERT dbo.DocumentStore(DocID, Title, MarketingSurveyGroup)
VALUES (2, 'Survey 2142', 'Men 25 - 35');
GO
```

При выборе всех столбцов таблицы возвращается обычный результирующий набор.

```
SELECT DocID, Title, ProductionSpecification, ProductionLocation,
MarketingSurveyGroup
FROM DocumentStore ;
```

Ниже приводится результирующий набор.

| DocID | Title | ProductionSpecification | ProductionLocation | MarketingSurveyGroup |
|-------|-------|-------------------------|--------------------|----------------------|
|-------|-------|-------------------------|--------------------|----------------------|

|   |             |         |      |           |
|---|-------------|---------|------|-----------|
| 1 | Tire Spec 1 | AXZZ217 | 27   | NULL      |
| 2 | Survey 2142 | NULL    | NULL | Men 25-35 |

Поскольку производственному отделу не нужны маркетинговые данные, должен быть предоставлен список столбцов, содержащих только необходимые данные, как это показано в следующем запросе.

```
SELECT DocID, Title, ProductionSpecification, ProductionLocation
FROM DocumentStore
WHERE ProductionSpecification IS NOT NULL ;
```

Ниже приводится результирующий набор.

| DocID | Title | ProductionSpecification | ProductionLocation |
|-------|-------|-------------------------|--------------------|
|-------|-------|-------------------------|--------------------|

1 Tire Spec 1 AXZZ217

27

## Использование наборов столбцов

В таблицах, использующих разреженные столбцы, можно назначить набор столбцов, который будет возвращать все разреженные столбцы в таблице. Набор столбцов — это нетипизированное XML-представление, которое объединяет на выходе все разреженные столбцы таблицы в структурированном виде. Набор столбцов похож на вычисляемые столбцы тем, что набор столбцов физически не хранится в таблице. Набор столбцов отличается от вычисляемого столбца тем, что он может быть напрямую обновлен.

Наборы столбцов следует использовать в том случае, если в таблице существует большое число столбцов и работать с ними по отдельности неудобно. У приложений может возрасти производительность, если они будут выбирать и вставлять данные в таблицы, имеющие много столбцов, с помощью наборов столбцов. Однако производительность наборов столбцов может уменьшиться, если для столбцов в таблице было определено большое количество индексов. Это происходит из-за увеличения объема памяти, необходимого для плана выполнения.

Чтобы определить набор столбцов, в инструкции [CREATE TABLE](#) или [ALTER TABLE](#) следует использовать ключевые слова <имя\_набора\_столбцов> FOR ALL\_SPARSE\_COLUMNS.

### Рекомендации по использованию наборов столбцов

При использовании наборов столбцов следует учитывать следующие рекомендации.

- Разреженные столбцы и набор столбцов могут быть созданы в рамках одной и той же инструкции.
- Набор столбцов нельзя изменять. Чтобы изменить набор столбцов, его нужно удалить и создать повторно.
- Набор столбцов не может быть добавлен в таблицу, если в ней уже содержатся разреженные столбцы.
- Набор столбцов может быть добавлен в таблицу, если в ней нет разреженных столбцов. Если впоследствии в таблицу будут добавлены разреженные столбцы, они появятся в наборе столбцов.
- В таблице может содержаться только один набор столбцов.
- Набор столбцов является дополнительной функцией, он не требуется для использования разреженных столбцов.
- Для набора столбцов нельзя определить ограничения или значения по умолчанию.
- Вычисляемые столбцы не могут содержать столбцы набора столбцов.
- Распределенные запросы не поддерживаются в таблицах, содержащих наборы столбцов.
- Репликация не поддерживает наборы столбцов.
- Система отслеживания измененных данных не поддерживает наборы столбцов.
- Набор столбцов не может быть частью никакого вида индексов. Это касается XML-индексов, полнотекстовых индексов и индексированных представлений. Набор столбцов не может быть добавлен как включенный столбец в любой

- индекс.
- Набор столбцов не может быть использован в критерии фильтра фильтруемого индекса или статистике фильтрации.
- Если представление содержит набор столбцов, в представлении он будет отображен как XML-столбец.
- Набор столбцов не может быть включен в определение индексированного представления.
- Секционированные представления, включающие таблицы, в которых содержатся наборы столбцов, могут быть обновлены, если секционированные представления упоминают разреженные столбцы по именам. Секционированное представление не может быть обновлено, если оно ссылается на набор столбцов.
- Не допускается использование уведомлений о запросах, ссылающихся на наборы столбцов.
- Предел размера XML-данных — 2 ГБ. Если сумма данных в строке во всех разреженных столбцах, содержащих значения, отличные от значений NULL, превышает этот предел, запрос или операция DML выдаст ошибку.
- Сведения о данных, возвращаемых функцией COLUMNS\_UPDATED, см. в разделе [Использование разреженных столбцов](#).

### Рекомендации по выбору данных из набора столбцов

Следует учитывать следующие рекомендации при выборе данных из набора столбцов.

- Фактически, набор столбцов — это тип обновляемого, вычисляемого XML-столбца, в котором набор базовых реляционных столбцов собирается в единое XML-представление. Набор столбцов поддерживает только свойство ALL\_SPARSE\_COLUMNS. Это свойство используется для сбора всех значений, отличных от значения NULL, из всех разреженных столбцов в определенной строке.
- В редакторе таблиц среды Среда SQL Server Management Studio наборы столбцов отображаются как изменяемые XML-поля. Наборы столбцов определяются с помощью следующего формата:

```
<column_name_1>value1</column_name_1><column_name_2>value2</column_name_2>...
```

Далее приводятся примеры значений набора столбцов:

- `<sparseProp1>10</sparseProp1><sparseProp3>20</sparseProp3>`
- `<DocTitle>Bicycle Parts List</DocTitle><Region>West</Region>`
- Разреженные столбцы, содержащие значения NULL, не включаются в XML-представление набора столбцов.

#### Внимание!

Добавление набора столбцов изменяет поведение запросов SELECT \*. Запрос будет возвращать набор столбцов как XML-столбец, а не как отдельные разреженные столбцы. Разработчики схем и приложений должны учитывать это, чтобы не нарушить работу существующих приложений.

### Вставка или изменение данных в наборе столбцов

Управлять данными в разреженных столбцах можно с помощью имен индивидуальных столбцов либо ссылаясь на имя набора столбцов и указывая значения набора столбцов, используя XML-формат набора столбцов. Разреженные столбцы могут быть расположены в XML-столбце в любом порядке.

```
CREATE TABLE t (i int SPARSE, cs xml column_set FOR ALL_SPARSE_COLUMNS)
GO
INSERT t(cs) VALUES ('<i/>')
GO
SELECT i FROM t
GO
```

## Использование типа данных `sql_variant`

| Тип данных                         | localeID*   | sqlCompareOptions | sqlCollationVersion | SqlSortId   | Максимальная длина | Точность      | Масштаб     |
|------------------------------------|-------------|-------------------|---------------------|-------------|--------------------|---------------|-------------|
| char, varchar, binary              | -1          | 'Default'         | 0                   | 0           | 8000               | Неприменимо** | Неприменимо |
| nvarchar                           | -1          | 'Default'         | 0                   | 0           | 4000               | Неприменимо   | Неприменимо |
| decimal, float, real               | Неприменимо | Неприменимо       | Неприменимо         | Неприменимо | Неприменимо        | 18            | 0           |
| integer, bigint, tinyint, smallint | Неприменимо | Неприменимо       | Неприменимо         | Неприменимо | Неприменимо        | Неприменимо   | Неприменимо |
| datetime                           | Неприм      | Неприм            | Неприм              | Неприм      | Неприм             | Неприм        | 7           |

|                               |             |             |             |             |             |             |             |
|-------------------------------|-------------|-------------|-------------|-------------|-------------|-------------|-------------|
| 2                             | енимо       | енимо       | енимо       | енимо       | енимо       | енимо       |             |
| datetime offset               | Неприменимо | Неприменимо | Неприменимо | Неприменимо | Неприменимо | Неприменимо | 7           |
| datetime, date, smalldatetime | Неприменимо | Неприменимо | Неприменимо | Неприменимо | Неприменимо | Неприменимо | Неприменимо |
| money, smallmoney             | Неприменимо | Неприменимо | Неприменимо | Неприменимо | Неприменимо | Неприменимо | Неприменимо |
| time                          | Неприменимо | Неприменимо | Неприменимо | Неприменимо | Неприменимо | Неприменимо | 7           |

\* значение localeID, равное -1, означает локаль по умолчанию. Локаль английского языка — 1033.

\*\* Неприменимо — во время операции выбора из набора столбцов нет никаких выходных значений для этих атрибутов. Формируется ошибка, если в XML-представлении, предоставленном для набора столбцов в операции вставки или обновления, вызывающий указал значение для этого атрибута.

## Безопасность

Модель безопасности набора столбцов работает схожим образом с моделью безопасности между таблицами и столбцами. Наборы столбцов могут быть визуализированы как минитаблица; операции выбора для данной минитаблицы имеют вид SELECT \*. Однако связь между набором столбцов и разреженными столбцами — это связь группирования, а не просто контейнер. Модель безопасности проверяет безопасность столбцов в наборе столбцов и выполняет операции DENY над базовыми разреженными столбцами. Далее приводятся дополнительные характеристики модели безопасности.

- Права доступа могут быть предоставлены и отменены на столбец в наборе столбцов так же, как и на любой другой столбец в таблице.
- Выполнение инструкции GRANT или REVOKE для разрешений SELECT, INSERT, UPDATE, DELETE и REFERENCES для столбца в наборе столбцов не распространяется на базовые столбцы-участники этого набора. Оно применяется только к столбцу в наборе столбцов. Разрешение DENY для набора столбцов распространяется на базовые разреженные столбцы таблицы.
- Чтобы выполнять инструкции SELECT, INSERT, UPDATE и DELETE над столбцами в наборе столбцов, пользователь должен иметь необходимые разрешения на столбец набора столбцов, а также соответствующее разрешение на все разреженные столбцы в таблице. Поскольку набор столбцов представляет все разреженные столбцы в таблице, пользователь должен обладать разрешением на все разреженные столбцы, включая и те, которые не будут изменены.
- Выполнение инструкции REVOKE над разреженным столбцом или набором столбцов устанавливает для него параметры безопасности, заданные по

умолчанию для его родительского объекта.

## Примеры

В следующих примерах в таблице документа содержится обычный набор столбцов DocID и Title. Производственной группе необходимы столбцы ProductionSpecification и ProductionLocation для всех рабочих документов. Группе сбыта необходим столбец MarketingSurveyGroup для документов сбыта.

### А. Создание таблицы с набором столбцов

В следующем примере создается таблица, в которой используются разреженные столбцы и содержится набор столбцов SpecialPurposeColumns. В этом примере в таблицу вставляются две строки, а затем из таблицы выбираются данные.

#### Примечание

Эта таблица насчитывает лишь пять столбцов, что упрощает ее отображение и чтение.

```
USE AdventureWorks2008R2;  
GO
```

```
CREATE TABLE DocumentStoreWithColumnSet  
(DocID int PRIMARY KEY,  
Title varchar(200) NOT NULL,  
ProductionSpecification varchar(20) SPARSE NULL,  
ProductionLocation smallint SPARSE NULL,  
MarketingSurveyGroup varchar(20) SPARSE NULL,  
MarketingProgramID int SPARSE NULL,  
SpecialPurposeColumns XML COLUMN_SET FOR ALL_SPARSE_COLUMNS);  
GO
```

### Б. Вставка данных в таблицу с использованием имен разреженных столбцов

В следующих примерах в таблицу, созданную в примере А, вставляются две строки. В примерах используются имена разреженных столбцов; набор столбцов не упоминается.

```
INSERT DocumentStoreWithColumnSet (DocID, Title, ProductionSpecification,  
ProductionLocation)  
VALUES (1, 'Tire Spec 1', 'AXZZ217', 27)  
GO
```

```
INSERT DocumentStoreWithColumnSet (DocID, Title, MarketingSurveyGroup)  
VALUES (2, 'Survey 2142', 'Men 25 - 35')  
GO
```

### В. Вставка данных в таблицу с использованием имени набора столбцов

В следующем примере в таблицу, созданную в примере А, вставляется третья строка. В этот раз имена разреженных столбцов не используются. Вместо этого используется имя набора столбцов, а операция вставки предоставляет значения для двух из четырех разреженных столбцов в формате XML.

```
INSERT DocumentStoreWithColumnSet (DocID, Title, SpecialPurposeColumns)
```

```
VALUES (3, 'Tire Spec 2',  
'<ProductionSpecification>AXW9R411</ProductionSpecification><ProductionLocation>  
38</ProductionLocation>')  
GO
```

#### Г. Рассмотрение результатов для набора столбцов при выполнении инструкции SELECT \*

В следующем примере из таблицы, содержащей набор столбцов, выбираются все столбцы. Возвращается XML-столбец, содержащий сочетание значений разреженных столбцов. Разреженные столбцы не возвращаются индивидуально.

```
SELECT * FROM DocumentStoreWithColumnSet ;
```

Ниже приводится результирующий набор.

DocID Title SpecialPurposeColumns

1 Tire Spec 1

<ProductionSpecification>AXZZ217</ProductionSpecification><ProductionLocation>27</ProductionLocation>

2 Survey 2142 <MarketingSurveyGroup>Men 25 - 35</MarketingSurveyGroup>

3 Tire Spec 2

<ProductionSpecification>AXW9R411</ProductionSpecification><ProductionLocation>38</ProductionLocation>

#### Д. Рассмотрение результатов выбора набора столбцов с использованием его имени

Поскольку производственному отделу не нужны маркетинговые данные, в этом примере для ограничения выходных данных добавляется предложение WHERE. В этом примере используется имя набора столбцов.

```
SELECT DocID, Title, SpecialPurposeColumns  
FROM DocumentStoreWithColumnSet  
WHERE ProductionSpecification IS NOT NULL ;  
Ниже приводится результирующий набор.
```

DocID Title SpecialPurposeColumns

1 Tire Spec 1

<ProductionSpecification>AXZZ217</ProductionSpecification><ProductionLocation>27</ProductionLocation>

3 Tire Spec 2

<ProductionSpecification>AXW9R411</ProductionSpecification><ProductionLocation>38</ProductionLocation>

#### Е. Рассмотрение результатов выбора разреженных столбцов с использованием их имен

Несмотря на то, что таблица содержит набор столбцов, можно выполнять запросы из таблицы с использованием имен отдельных столбцов. Это показано в следующем примере.

```
SELECT DocID, Title, ProductionSpecification, ProductionLocation
FROM DocumentStoreWithColumnSet
WHERE ProductionSpecification IS NOT NULL ;
```

Ниже приводится результирующий набор.

| DocID | Title | ProductionSpecification | ProductionLocation |
|-------|-------|-------------------------|--------------------|
|-------|-------|-------------------------|--------------------|

|   |             |         |    |
|---|-------------|---------|----|
| 1 | Tire Spec 1 | AXZZ217 | 27 |
|---|-------------|---------|----|

|   |             |          |    |
|---|-------------|----------|----|
| 3 | Tire Spec 2 | AXW9R411 | 38 |
|---|-------------|----------|----|

## Ж. Обновление таблицы с помощью набора столбцов

В следующем примере третья запись обновляется новыми значениями для обоих разреженных столбцов, использующихся в этой строке.

```
UPDATE DocumentStoreWithColumnSet
SET SpecialPurposeColumns =
'<ProductionSpecification>ZZ285W</ProductionSpecification><ProductionLocation>38
</ProductionLocation>'
WHERE DocID = 3 ;
GO
```

### Важно!

Инструкция UPDATE, использующая набор столбцов, обновляет все разреженные столбцы в таблице. Для всех разреженных столбцов, которые не были упомянуты, устанавливается значение NULL.

В следующем примере обновляется третья запись, однако значение указывается только для одного из двух заполненных столбцов. Второй столбец, ProductionLocation, не включен в инструкцию UPDATE, и для него устанавливается значение NULL.

```
UPDATE DocumentStoreWithColumnSet
SET SpecialPurposeColumns =
'<ProductionSpecification>ZZ285W</ProductionSpecification>'
WHERE DocID = 3 ;
GO
```

## Создание и изменение ограничений PRIMARY KEY

При создании таблицы в ее определении можно указать одно ограничение PRIMARY KEY. Если таблица уже существует, ограничение PRIMARY KEY может быть добавлено только в том случае, если для таблицы еще не определено ни одного такого ограничения. У таблицы может быть только одно ограничение PRIMARY KEY.

Если ограничение PRIMARY KEY уже существует, его можно изменить или удалить. Например, может потребоваться, чтобы ограничение PRIMARY KEY ссылалось на другие столбцы, или нужно изменение порядка столбцов, имени индекса, параметров кластеризации или коэффициента заполнения. Но длину столбца, определенного с ограничением PRIMARY KEY, изменить нельзя.



**Примечание**

Чтобы изменить ограничение PRIMARY KEY, необходимо сначала удалить существующее ограничение PRIMARY KEY, а затем создать новое с другим определением.

Если ограничение PRIMARY KEY добавляется к существующему столбцу или столбцам таблицы, компонент Database Engine проверяет данные и метаданные этих столбцов, чтобы проверить их соответствие следующим правилам.

- Столбцы не должны допускать значений NULL.

Столбцы ограничения PRIMARY KEY, указанного при создании таблицы, неявно преобразуются в NOT NULL. Разреженный столбец нельзя использовать как часть первичного ключа, так как разреженные столбцы должны допускать значения NULL.

- В них не должно быть повторяющихся значений.

Если ограничение PRIMARY KEY добавляется к столбцу, который содержит повторяющиеся значения или допускает значения NULL, компонент Database Engine возвращает ошибку и ограничение не создается.

Ограничение PRIMARY KEY не может быть добавлено к столбцам, нарушающим перечисленные правила.

Компонент Database Engine автоматически создает уникальный индекс, чтобы гарантировать уникальность ограничения PRIMARY KEY. Если для таблицы не существует кластеризованный индекс и явно не указан некластеризованный, то для ограничения PRIMARY KEY создается уникальный кластеризованный индекс.

Ограничение PRIMARY KEY не может быть удалено в следующих случаях.

- Если на него ссылается ограничение FOREIGN KEY из другой таблицы. Сначала должно быть удалено ограничение FOREIGN KEY.
- К таблице применен индекс PRIMARY XML.

Создание ограничения PRIMARY KEY при создании таблицы

- [Инструкция CREATE TABLE \(Transact-SQL\)](#)

Создание ограничения PRIMARY KEY в существующей таблице

- [ALTER TABLE \(Transact-SQL\)](#)
- [Как создать первичные ключи \(визуальные инструменты для баз данных\)](#)
- [Как удалить первичные ключи \(визуальные инструменты для баз данных\)](#)

Получение сведений об ограничениях PRIMARY KEY

- [sys.key\\_constraints \(Transact-SQL\)](#)

## Создание и изменение ограничений FOREIGN KEY

Ограничение FOREIGN KEY можно определить при создании таблицы. Если таблица уже существует, ограничение FOREIGN KEY можно добавить, в этом случае оно присоединяется к существующим ограничениям PRIMARY KEY или UNIQUE этой или другой таблицы. Таблица может содержать несколько ограничений FOREIGN KEY.

Существующее ограничение FOREIGN KEY можно изменить или удалить. Возможно, потребуется, чтобы ограничение FOREIGN KEY ссылалось на другие столбцы. Однако

длину столбца, определяющего ограничение FOREIGN KEY, изменить нельзя.

#### Примечание

Чтобы изменить ограничение FOREIGN KEY, его необходимо удалить и повторно создать с новым определением.

Ограничение FOREIGN KEY необходимо удалить, чтобы снять требования ссылочной целостности между столбцами внешнего ключа и соответствующими столбцами первичного ключа или ограничения UNIQUE в другой таблице.

Создание ограничения FOREIGN KEY вместе с таблицей

[Инструкция CREATE TABLE \(Transact-SQL\)](#)

Создание ограничения FOREIGN KEY в существующей таблице

[ALTER TABLE \(Transact-SQL\)](#)

[Как создать связь между таблицами \(визуальные инструменты для баз данных\)](#)

Удаление ограничения FOREIGN KEY

[ALTER TABLE \(Transact-SQL\)](#)

### ***Принудительное применение ограничения FOREIGN KEY при помощи параметра WITH NOCHECK***

Когда ограничение FOREIGN KEY добавляется в существующую таблицу, компонент Database Engine проверяет данные столбцов, чтобы убедиться, что все значения, кроме NULL, существуют в соответствующих столбцах ограничения PRIMARY KEY или UNIQUE. Если указать параметр WITH NOCHECK, компонент Database Engine добавит новое ограничение, не проверяя, соответствуют ли ему данные столбца. Параметр WITH NOCHECK удобно использовать, если существующие данные столбца уже удовлетворяют новому ограничению FOREIGN KEY или бизнес-правило требует, чтобы ограничение вступало в силу только с данного момента времени.

Тем не менее при создании ограничения без проверки существующих данных необходимо соблюдать осторожность, поскольку в этом случае компонент Database Engine не сможет гарантировать ссылочную целостность таблицы.

Избежание проверки существующих данных при создании ограничения FOREIGN KEY

[ALTER TABLE \(Transact-SQL\)](#)

### ***Отключение ограничения FOREIGN KEY***

Существующие ограничения FOREIGN KEY можно при выполнении таких операций, как INSERT или UPDATE, и во время обработки репликации.

- Инструкции INSERT и UPDATE

Если отключить ограничение FOREIGN KEY, данные таблицы будут изменяться без проверки на соответствие ограничениям. Ограничение FOREIGN KEY можно отключить во время выполнения инструкций INSERT или UPDATE, если новые данные не удовлетворяют ограничению или оно должно распространяться только

на те данные, которые содержатся в базе данных.

#### Примечание

Любые каскадные действия, определенные для соответствующего первичного ключа, не будут выполняться над строками, для которых внешний ключ отключен.

- **Обработка репликации**

Отключите ограничение FOREIGN KEY во время репликации, если оно должно выполняться только в исходной базе данных. При репликации таблицы определение и данные таблицы копируются из исходной в целевую таблицу. Если не отключить ограничение FOREIGN KEY, которое должно выполняться только в исходной базе данных, во время репликации, оно может препятствовать копированию данных в целевую базу данных. Дополнительные сведения см. в разделе [Управление ограничениями, идентификаторами и триггерами с помощью параметра «NOT FOR REPLICATION»](#).

Отключение ограничения FOREIGN KEY во время выполнения инструкций INSERT и UPDATE

[ALTER TABLE \(Transact-SQL\)](#)

[Как отключить ограничение внешнего ключа при помощи инструкций INSERT и UPDATE \(визуальные инструменты для баз данных\)](#)

Отключены ограничения FOREIGN KEY во время репликации

[ALTER TABLE \(Transact-SQL\)](#)

[Как отключить ограничения внешнего ключа для репликации \(визуальные инструменты для баз данных\)](#)

Получение сведений об ограничениях FOREIGN KEY

[sys.foreign\\_keys \(Transact-SQL\)](#)

Получение сведений о столбцах, составляющих ограничение FOREIGN KEY

[sys.foreign\\_key\\_columns \(Transact-SQL\)](#)

## Создание и изменение ограничений UNIQUE

Ограничение UNIQUE может быть создано как часть определения таблицы при ее создании. Ограничение UNIQUE может быть добавлено к уже существующей таблице при условии, что столбец или комбинация столбцов, на которых оно основывается, содержат только уникальные значения. Таблица может содержать несколько ограничений UNIQUE.

Если ограничение UNIQUE уже существует, оно может быть изменено или удалено. Например, может понадобиться связать ограничение UNIQUE с другими столбцами либо изменить тип кластеризации индекса.

#### Примечание

Чтобы изменить ограничение UNIQUE, необходимо вначале удалить существующее ограничение, а затем создать его повторно с помощью нового определения.

При добавлении ограничения UNIQUE на уже существующий столбец или группу столбцов в таблице, компонент Database Engine по умолчанию проверяет уникальность всех существующих значений в указанных столбцах. При попытке добавить ограничение UNIQUE к столбцу, содержащему повторяющиеся значения, компонент Database Engine возвращает ошибку, а ограничение не добавляется.

Компонент Database Engine автоматически создает индекс UNIQUE, что обеспечивает выполнение требований уникальности значений для ограничений UNIQUE. Поэтому, при попытке вставки в таблицу строки с повторяющимися данными, компонент Database Engine выдает сообщение об ошибке, в котором сообщается о нарушении ограничения UNIQUE, а строка в таблицу не вставляется. Для обеспечения выполнения ограничения UNIQUE по умолчанию создается уникальный некластеризованный индекс, если явно не указано создание кластеризованного индекса.

В приведенном ниже фрагменте инструкции CREATE TABLE показано определение столбца Name и на этот столбец задано ограничение UNIQUE, гарантирующее уникальность значений.

```
Name nvarchar(100) NOT NULL
```

```
UNIQUE NONCLUSTERED
```

Чтобы снять требования уникальности, предъявляемые значениям в столбцах таблицы, включенной в ограничение UNIQUE, необходимо удалить это ограничение.

Ограничение UNIQUE не может быть удалено, если связанный с ним столбец используется в качестве полнотекстового ключа таблицы.

Создание ограничения UNIQUE при создании таблицы

- [Инструкция CREATE TABLE \(Transact-SQL\)](#)

Создание ограничения UNIQUE в существующей таблице

- [ALTER TABLE \(Transact-SQL\)](#)

Удаление ограничения UNIQUE

- [ALTER TABLE \(Transact-SQL\)](#)

Получение сведений об ограничении UNIQUE

- [sys.key\\_constraints \(Transact-SQL\)](#)

## Создание и изменение ограничений CHECK

Ограничения CHECK можно создавать как часть определения таблицы при ее создании. Если таблица уже существует, можно добавить ограничение CHECK. Таблицы и столбцы могут содержать несколько ограничений CHECK.

Если ограничение CHECK уже существует, его можно изменить или удалить. Например, может возникнуть необходимость изменить выражение, используемое в ограничении CHECK для столбца таблицы.

### Примечание

Чтобы изменить ограничение CHECK, нужно, прежде всего удалить существующее

ограничение CHECK и повторно создать его с новым определением.

В следующем примере Transact-SQL показано создание таблицы и последующее изменение этой таблицы путем добавления ограничения CHECK для столбца CreditRating.

```
IF OBJECT_ID ('dbo.Vendors', 'U') IS NOT NULL
DROP TABLE dbo.Vendors;
GO
CREATE TABLE dbo.Vendors
    (VendorID int PRIMARY KEY, VendorName nvarchar (50),
    CreditRating tinyint)
GO
ALTER TABLE dbo.Vendors ADD CONSTRAINT CK_Vendor_CreditRating
    CHECK (CreditRating >= 1 AND CreditRating <= 5)
```

Удалите ограничения CHECK для устранения ограничений на приемлемые значения данных в столбце или столбцах, включенных в выражение ограничения.

Для создания ограничения CHECK в процессе создания таблицы

- [Инструкция CREATE TABLE \(Transact-SQL\)](#)

Для создания ограничения CHECK в существующей таблице

- [ALTER TABLE \(Transact-SQL\)](#)
- [Как привязать новое проверочное ограничение к таблице или столбцу \(визуальные инструменты для баз данных\)](#)

Для удаления ограничения CHECK

- [ALTER TABLE \(Transact-SQL\)](#)
- [Как удалить проверочное ограничение \(визуальные инструменты для баз данных\)](#)

### ***Форсирование ограничения CHECK с помощью параметра WITH NOCHECK***

При добавлении ограничения CHECK к существующей таблице ограничение CHECK можно применять только к новым данным или к существующим данным. По умолчанию, ограничение CHECK применяется как к существующим данным, так и к новым данным. Параметр WITH NOCHECK инструкции ALTER TABLE используется для применения нового ограничения только к вновь добавляемым данным. Этот параметр полезно использовать в случаях, когда существующие данные уже соответствуют новому ограничению CHECK или когда бизнес-правило требует форсированного применения ограничения, начиная с данного момента.

Например, старое ограничение может предусматривать использование не более чем пятизначных почтовых кодов, а новое ограничение может требовать применения девятизначных почтовых кодов. Старые данные с пятизначными почтовыми кодами все еще действительны и будут сосуществовать с новыми данными, которые содержат девятизначные почтовые коды. Следовательно, проверке на соответствие новому ограничению должны подвергаться только новые данные.

Однако следует проявлять осторожность при добавлении ограничений без проверки существующих данных, поскольку в этом случае не используются средства управления компонентом Database Engine, обеспечивающие соблюдение правил целостности для таблиц.

Для исключения проверки существующих данных при создании ограничения CHECK

- [ALTER TABLE \(Transact-SQL\)](#)
- [Как проверить существующие данные при создании проверочного ограничения \(визуальные инструменты для баз данных\)](#)

### **Отключение ограничений CHECK**

Существует возможность отключения ограничений CHECK при выполнении некоторых операций, таких как операции INSERT, операции UPDATE и операции по обработке репликации.

- Инструкции INSERT и UPDATE  
Отключение ограничения CHECK позволяет изменять данные в таблице без проверки на предмет соответствия этому ограничению. Отключите ограничение CHECK при выполнении инструкций INSERT и UPDATE, если новые данные будут нарушать это ограничение или если ограничение должно применяться только по отношению к данным, уже внесенным в базу данных.
- Обработка репликации  
Отключите ограничение CHECK в процессе репликации, если ограничение применимо только к базе данных-источнику. При репликации таблицы определение таблицы и данные копируются из базы данных-источника в целевую базу данных. Как правило, но необязательно, эти две базы данных располагаются на отдельных серверах. Если ограничения CHECK, применимые только к базе данных-источнику, не отключены, они могут создавать ненужные помехи для введения новых сведений в целевую базу данных. Дополнительные сведения см. в разделе [Управление ограничениями, идентификаторами и триггерами с помощью параметра «NOT FOR REPLICATION»](#).

Для отключения ограничения CHECK при выполнении инструкций INSERT и UPDATE

- [ALTER TABLE \(Transact-SQL\)](#)
- [Как отключить проверочные ограничения при помощи инструкций INSERT и UPDATE \(визуальные инструменты для баз данных\)](#)

Для отключения ограничения CHECK при выполнении репликации

- [ALTER TABLE \(Transact-SQL\)](#)
- [Как отключить проверочные ограничения для репликации \(визуальные инструменты для баз данных\)](#)

Для получения сведений об ограничениях CHECK

- [sys.check\\_constraints \(Transact-SQL\)](#)

## **Изменение и создание определений DEFAULT**

Можно создать определение DEFAULT в определении таблицы при ее создании. Если таблица уже существует, можно добавить к ней определение DEFAULT. Каждый столбец таблицы может содержать одно определение DEFAULT.

Если определение DEFAULT уже существует, можно изменить его или удалить.

Например можно изменить значение, которое будет помещаться в столбец, если при вставке значение не указано.

Для изменения определения DEFAULT необходимо сначала удалить существующее определение DEFAULT, а затем создать его повторно в новом определении.

Для столбцов, определения которых включают следующие элементы или свойства, создать определение DEFAULT невозможно:

- Тип данных timestamp.
- Разреженный столбец — поскольку разреженный столбец должен допускать значения NULL.
- Свойство IDENTITY или ROWGUIDCOL.
- Существующее определение DEFAULT или объект DEFAULT.

#### Примечание

Заданное по умолчанию значение должно быть совместимо с типом данных столбца, к которому применяется определение DEFAULT. Например, заданное по умолчанию значение столбца типа int должно быть целым числом, а не символьной строкой.

Когда определение DEFAULT добавляется к существующему в таблице столбцу, по умолчанию Database Engine использует это определение лишь для новых добавляемых в таблицу данных. Существующие данные, которые вносились во время действия предыдущего определения DEFAULT, остаются без изменений. Однако при добавлении нового столбца к существующей таблице, можно указать, чтобы вместо значения NULL Database Engine поместил значение по умолчанию (указанное в определении DEFAULT) в новый столбец во всех существующих в таблице строках.

После удаления определения DEFAULT Database Engine будет помещать в соответствующий столбец добавляемых строк значение NULL, а не значение по умолчанию, когда при вставке значение столбца не указано. Однако уже находящиеся в таблице данные не изменяются.

Создание определения DEFAULT для столбца при создании таблицы

- [Инструкция CREATE TABLE \(Transact-SQL\)](#)

Создание или удаление определения DEFAULT для столбца существующей таблицы

- [ALTER TABLE \(Transact-SQL\)](#)
- [Как присвоить столбцам значения по умолчанию \(визуальные инструменты для баз данных\)](#)

[Свойства столбца таблицы \(среда SQL Server Management Studio\)](#)

- Удаление объекта DEFAULT
- [DROP DEFAULT \(Transact-SQL\)](#)

Получение сведений об определении DEFAULT

- [sys.default\\_constraints \(Transact-SQL\)](#)

## Создание и изменение столбцов идентификаторов

Для создания автоматически увеличивающегося идентификационного номера в таблице можно использовать столбец идентификаторов. Для каждой таблицы можно создать только один столбец идентификаторов и один столбец идентификаторов GUID.



### Свойство **IDENTITY**

Реализовать столбцы идентификаторов можно при помощи свойства **IDENTITY**. Это позволяет разработчику указать как количество идентификаторов для первой строки, вставляемой в таблицу (свойство **Начальное значение идентификатора**), так и увеличение (свойство **Шаг приращения идентификатора**), добавляемое к начальному значению для определения последующих номеров идентификаторов. При вставке значений в таблицу со столбцом идентификаторов компонента Database Engine автоматически формирует следующее значение идентификатора, добавляя значение шага приращения идентификатора к начальному значению. При добавлении столбцов идентификатора к существующим таблицам идентификационные номера добавляются в существующие строки таблицы. При этом начальное значение и значения приращения применяются в том же порядке, в котором строки были вставлены. Номера идентификаторов также формируются для всех новых строк, которые добавляются. Нельзя изменить существующий столбец таблицы, добавив в него свойство **IDENTITY**.

Если для определения столбца идентификаторов используется свойство **IDENTITY**, необходимо помнить следующее:

- таблица может содержать только один столбец со свойством **IDENTITY**, причем такой столбец должен иметь тип данных **decimal**, **int**, **numeric**, **smallint**, **bigint** или **tinyint**;
- можно указать начальное значение и шаг. Значение по умолчанию для обоих равно 1;
- столбец идентификатора не должен допускать значений **NULL** и содержать определений или объектов по умолчанию;
- ссылку на столбец в списке выборки можно создать с помощью указания ключевого слова **\$IDENTITY** после свойства **IDENTITY**. Сослаться на столбец можно также при помощи имени;
- функцию **OBJECTPROPERTY** можно использовать для определения наличия в таблице столбца со свойством **IDENTITY**, а функция **COLUMNPROPERTY** может быть использована для определения имени столбца со свойством **IDENTITY**;
- чтобы отключить свойство **IDENTITY** для столбца, можно использовать инструкцию **SET IDENTITY\_INSERT**, активируя значения, которые должны быть вставлены явным образом.

#### Примечание

Если в таблице, в которой часто производятся удаления, имеется столбец идентификаторов, между значениями идентификатора могут возникнуть промежутки. Удаленные значения идентификаторов повторно не используются. Чтобы избежать возникновения таких промежутков, не используйте свойство **IDENTITY**. Вместо этого можно создать триггер, который определяет новое значение идентификатора на основе существующих значений в столбцах идентификатора по мере вставки строк.

### Глобальные уникальные идентификаторы

Несмотря на то, что свойство **IDENTITY** автоматизирует нумерацию строк в рамках одной таблицы, разные таблицы, каждая со своим столбцом идентификаторов, могут создавать одинаковые значения. Это обусловлено тем, что уникальность свойства **IDENTITY** гарантирована только в рамках таблицы, в которой оно использовано. Если в



приложении нужно сформировать столбец идентификатора, уникальный для всей базы данных или всех баз данных во всех сетях компьютеров в мире, используется тип данных `uniqueidentifier` и функция [NEWID](#) или [NEWSEQUENTIALID\(\)](#). Кроме того, можно применить свойство `ROWGUIDCOL`, чтобы указать, что новый столбец является столбцом идентификаторов GUID строк. В отличие от столбцов, определенных с помощью свойства `IDENTITY`, компонент Database Engine не создает автоматически значения для столбцов с типом данных `uniqueidentifier`. Чтобы вставить глобальное уникальное значение, создайте для столбца определение `DEFAULT`, которое использует функцию `NEWID` или `NEWSEQUENTIALID` для формирования глобального уникального значения. Дополнительные сведения см. в разделе [Использование данных uniqueidentifier](#).

На столбец со свойством `ROWGUIDCOL` можно ссылаться в списке выбора, используя ключевое слово `$ROWGUID`. Это аналогично ссылке на столбец `IDENTITY` при помощи ключевого слова `$IDENTITY`; в таблице может содержаться только один столбец `ROWGUIDCOL`, который должен иметь тип данных `uniqueidentifier`.

Функцию [OBJECTPROPERTY \(Transact-SQL\)](#) можно использовать для определения наличия в таблице столбца со свойством `ROWGUIDCOL`, а функция [COLUMNPROPERTY \(Transact-SQL\)](#) может быть использована для определения имени столбца со свойством `ROWGUIDCOL`.

В следующем примере создается таблица с указанием столбца **`uniqueidentifier`** в качестве первичного ключа. Пример использует функцию `NEWSEQUENTIALID()` с ограничением `DEFAULT` для задания значений для новых строк. К столбцу **`uniqueidentifier`** применяется свойство `ROWGUIDCOL`, так что на столбец можно ссылаться с помощью ключевого слова `$ROWGUID`.

Transact-SQL

```
CREATE TABLE dbo.Globally_Unique_Data
    (guid uniqueidentifier CONSTRAINT Guid_Default DEFAULT NEWSEQUENTIALID()
    ROWGUIDCOL,
    Employee_Name varchar(60)
    CONSTRAINT Guid_PK PRIMARY KEY (guid) );
```

Создание нового столбца идентификаторов при создании таблицы

[Инструкция CREATE TABLE \(Transact-SQL\)](#)

Создание столбца идентификаторов в существующей таблице

[ALTER TABLE \(Transact-SQL\)](#)

Удаление столбца идентификаторов

[ALTER TABLE \(Transact-SQL\)](#)

[Как удалять столбцы из таблицы \(визуальные инструменты для баз данных\)](#)

Получение сведений о столбце идентификаторов

[sys.identity\\_columns \(Transact-SQL\)](#)

[IDENT\\_INCR \(Transact-SQL\)](#)

[IDENT\\_SEED \(Transact-SQL\)](#)

Проверка и исправление текущего значения идентификатора заданной таблицы

[DBCC CHECKIDENT \(Transact-SQL\)](#)

Установка нового начального значения

[DBCC CHECKIDENT \(Transact-SQL\)](#)

## Создание сжатых таблиц и индексов

SQL Server 2008 обеспечивает сжатие как строк, так и страниц для таблиц и индексов. Сжатие данных может быть настроено для следующих объектов базы данных.

- Для полной таблицы, хранящейся в виде кучи.
- Для полной таблицы, хранящейся в виде кластеризованного индекса.
- Для полного некластеризованного индекса.
- Для полного некластеризованного представления.
- Для секционированных таблиц и индексов параметры сжатия можно задать для каждой секции, разные секции объекта могут иметь различные параметры.

Параметры сжатия таблицы не применяются автоматически к ее некластеризованным индексам. Каждый индекс должен быть установлен индивидуально. Для системных таблиц сжатие недоступно. Таблицы и индексы могут быть сжаты, если для их создания были использованы инструкции [CREATE TABLE](#) и [CREATE INDEX](#). Чтобы изменить состояние сжатия таблицы, индекса или секции, используются инструкции [ALTER TABLE](#) или [ALTER INDEX](#).

### Примечание

Если существующие данные фрагментированы, уменьшить размер индекса можно, перестроив его, не используя сжатие. Коэффициент заполнения индекса будет применен во время перестройки индекса, что потенциально может привести к увеличению размера индекса. Дополнительные сведения см. в разделе [Коэффициент заполнения](#).

### Замечания по использованию сжатия строк и страниц

При использовании сжатия строк и страниц следует учитывать следующее.

- Сжатие доступно только в выпусках SQL Server 2008 Enterprise и Developer.
- С помощью сжатия можно хранить больше строк в странице, максимальный размер строки таблицы или индекса при этом не изменяется.
- Для таблицы нельзя включить сжатие, если сумма максимального размера строки и служебных данных сжатия превышает максимальный размер строки в 8060 байт. Например, таблица, в которой присутствуют столбцы c1 char(8000) и c2 char(53), не может быть сжата, поскольку добавление служебных данных сжатия приведет к превышению максимального размера строки. При использовании формата хранения vardecimal выполняется проверка размера строки (когда формат включен). При использовании сжатия строк и страниц проверка размера строки выполняется при первичном сжатии объекта, а также при всех последующих

вставках и изменениях строк. При использовании сжатия обеспечивается выполнение следующих двух правил.

- Обновление для типа с фиксированной длиной должно всегда завершаться успешно.
- Отключение сжатия данных должно всегда выполняться успешно. Даже если сжатая строка уместается на странице (что означает, что она имеет размер меньше 8060 байт), SQL Server не допустит выполнения обновлений, которые не поместятся в строке без сжатия.
- Если указан список секций, для каждой отдельной секции можно установить тип сжатия ROW, PAGE или NONE. Если список секций не был указан, для всех секций устанавливается свойство сжатия данных, указанное в инструкции. При создании индекса или таблицы для свойства сжатия устанавливается значение NONE, если не было указано другое значение. При изменении таблицы сохраняется существующее сжатие, если не было указано иное.
- При указании списка секций или секции вне диапазона формируется ошибка.
- Некластеризованные индексы не наследуют свойство сжатия таблицы. Чтобы сжать индексы, необходимо явно задать для них свойство сжатия. По умолчанию при создании индексов для них устанавливается режим сжатия NONE.
- При создании кластеризованного индекса в куче кластеризованный индекс наследует состояние сжатия кучи, если не указано другое состояние сжатия.
- Если для кучи было настроено сжатие уровня страницы, для страниц такое сжатие будет реализовываться только следующими методами.
  - Массовый импорт данных осуществляется при включенной оптимизации массовых операций.
  - Вставка данных с помощью синтаксиса INSERT INTO ... WITH(TABLOCK).
  - Перестройка таблицы с помощью инструкции ALTER TABLE ... REBUILD с параметром сжатия PAGE.
- В новых страницах, размещенных в куче в процессе выполнения операций DML, сжатие страниц не будет использоваться до тех пор, пока куча не будет перестроена. Перестройте кучу. Для этого удалите и повторно примените сжатие либо создайте и удалите кластеризованный индекс.
- Чтобы изменить параметры сжатия кучи, необходимо перестроить все некластеризованные индексы в таблице. Это обеспечивает наличие в них указателей на новые расположения в куче.
- Включить или отключить сжатие типа ROW или PAGE можно в оперативном или режиме вне сети. Включение сжатия для кучи является однопоточным для операции в сети.
- Чтобы включить или отключить сжатие строки или страницы, необходимо столько же места на диске, как и для создания или перестройки индекса. Для секционированных данных объем пространства, необходимый для включения или отключения сжатия, можно сократить, выполняя включение или отключение сжатия последовательно для каждой секции.
- Чтобы определить состояние сжатия секций в секционированной таблице, выполните запрос столбца data\_compression из представления каталога sys.partitions.
- При сжатии индексов страницы конечного уровня можно сжать как сжатием строк, так и сжатием страниц. Для страниц, расположенных не на конечном уровне, нельзя использовать сжатие страниц.
- Ввиду их размера, типы данных больших значений иногда хранятся отдельно от нормальных данных строк на страницах для особых целей. Сжатие данных недоступно для данных, хранящихся отдельно.

- Таблицы, для которых в SQL Server 2005 был реализован формат хранения vardecimal, сохраняют эту настройку и после обновления. К таблице, где используется формат хранения vardecimal, можно применить сжатие строк. Но поскольку сжатие строк является надмножеством формата хранения vardecimal, не существует причин сохранения данного формата. Для десятичных значений не происходит никакого дополнительного сжатия при сочетании формата хранения vardecimal и сжатия строк. Сжатие страниц можно применить к таблице, в которой присутствует формат хранения vardecimal, однако для столбцов данного формата дополнительного сжатия, скорее всего, не произойдет.

#### Примечание

SQL Server 2008 поддерживает формат хранения vardecimal, однако, поскольку сжатие уровня строк достигает тех же целей, данный формат является устаревшим. В будущей версии Microsoft SQL Server эта возможность будет удалена. Избегайте использования этой возможности в новых разработках и запланируйте изменение существующих приложений, в которых она применяется.

### Реализация сжатия

Сводку по реализации сжатия данных см. в разделах [Реализация сжатия строк](#), [Реализация сжатия страниц](#) и [Реализация сжатия Юникода](#).

### Ожидаемая экономия при сжатии

Определить результат сжатия таблицы или индекса можно с помощью хранимой процедуры [sp\\_estimate\\_data\\_compression\\_savings](#). Хранимая процедура `sp_estimate_data_compression_savings` доступна только в тех выпусках SQL Server, где поддерживается сжатие данных.

### Влияние сжатия на секционированные таблицы и индексы

При использовании сжатия данных в секционированных таблицах или индексах следует учитывать следующее.

- Разбиение диапазона  
При разбиении секций с помощью инструкции `ALTER PARTITION` обе секции наследуют атрибут сжатия данных исходной секции.
- Слияние диапазона  
При слиянии двух секций результирующая секция унаследует атрибут сжатия данных секции назначения.
- Переключение секций  
Для переключения секции свойство сжатия данных секции должно совпадать со свойством сжатия таблицы.
- Перестроение одной или всех секций  
Существуют две вариации синтаксиса, которые можно использовать для изменения сжатия секционированной таблицы или индекса.
  - Следующий синтаксис перестраивает только упоминаемую секцию:  

```
ALTER TABLE <table_name>  
REBUILD PARTITION = 1 WITH (DATA_COMPRESSION = <option>)
```
  - Следующий синтаксис перестраивает всю таблицу, используя существующий

режим сжатия для всех неупоминаемых секций:

```
ALTER TABLE <table_name>
REBUILD PARTITION = ALL
WITH (DATA_COMPRESSION = PAGE ON PARTITIONS(<range>),
... )
```

Для секционированных индексов действуют те же принципы, но используется инструкция ALTER INDEX.

- Удаление секционированного кластеризованного индекса

При удалении кластеризованного индекса соответствующие секции кучи сохраняют настройки сжатия данных, если только не была изменена схема секционирования. Если схема секционирования подверглась изменениям, все секции перестраиваются в распакованное состояние. Чтобы удалить кластеризованный индекс и изменить схему секционирования, необходимо выполнить следующие шаги.

1. Удалить кластеризованный индекс.
2. Изменить таблицу с помощью параметра REBUILD ... инструкции ALTER TABLE ..., в котором указывается режим сжатия.

Удаление кластеризованного индекса в режиме вне сети происходит очень быстро, поскольку удаляются только верхние уровни кластеризованных индексов. При удалении кластеризованного индекса в режиме в сети SQL Server должен перестроить кучу два раза — один для первого шага, один для второго.

### **Влияние сжатия на репликацию**

При использовании сжатия данных с репликацией следует учитывать следующее.

- Если агент моментальных снимков формирует первичный сценарий схемы, в новой схеме для таблицы и ее индексов будет использован один и тот же режим сжатия. Сжатие нельзя включить только для таблицы, но не для индекса.
- При репликации транзакций параметр схемы статьи определяет, какие из зависимых объектов или свойств должны быть добавлены в сценарий. Дополнительные сведения см. в разделе [sp\\_addarticle](#). Агент распространителя не проверяет наличие подписчиков низкого уровня при применении сценариев. При выборе репликации сжатия создание таблицы у подписчиков низкого уровня завершится неудачно. При наличии смешанной топологии не следует включать репликацию сжатия.
- Для репликации слиянием уровень совместимости публикаций переопределяет параметры схемы и определяет, какие из объектов схемы будут внесены в сценарий. Дополнительные сведения об уровне совместимости см. в разделе [Использование нескольких версий SQL Server в топологии репликации](#). При наличии смешанной топологии для уровня совместимости публикации следует установить значение версии подписчика низкого уровня, если поддержка новых параметров сжатия не является обязательной. Если поддержка необходима, после создания таблиц их сжатие следует осуществить на подписчике.

В следующей таблице показываются настройки репликации, управляющие сжатием в процессе репликации.

| Намерение пользователя | Выполнить репликацию схемы секционирования для таблицы или | Выполнить репликацию настроек сжатия | Действия со сценариями |
|------------------------|------------------------------------------------------------|--------------------------------------|------------------------|
|------------------------|------------------------------------------------------------|--------------------------------------|------------------------|

|                                                                                                                                                  | индекса |       |                                                                                             |
|--------------------------------------------------------------------------------------------------------------------------------------------------|---------|-------|---------------------------------------------------------------------------------------------|
| Выполнить репликацию схемы секционирования и включить сжатие на подписчике для этой секции.                                                      | True    | True  | Создать сценарии для схемы секционирования и для настроек сжатия.                           |
| Выполнить репликацию схемы секционирования, но не сжимать данные на подписчике.                                                                  | True    | False | Создать сценарий для схемы секционирования, но не создавать его для настроек сжатия секции. |
| Не выполнять репликацию схемы секционирования и не сжимать данные на подписчике.                                                                 | False   | False | Не создавать сценарии для настроек секций или сжатия.                                       |
| Выполнить сжатие таблицы на подписчике при условии, что на издательстве были сжаты все секции, но не выполнять репликацию схемы секционирования. | False   | True  | Проверить, что для всех секций включено сжатие. Создать сценарий сжатия на уровне таблицы.  |

### **Влияние сжатия на другие компоненты SQL Server**

Сжатие происходит в подсистеме хранилища, и данные предоставляются большинству других компонентов SQL Server в распакованном состоянии. Это ограничивает влияние сжатия на другие компоненты следующим.

- Операции массового импорта и экспорта  
При экспорте данных, даже в собственном формате, данные выводятся в распакованном формате строк. Поэтому размер экспортированного файла данных может значительно превысить размер исходных данных.  
При импорте данных, если для целевой таблицы было включено сжатие, данные будут преобразованы подсистемой хранилища в сжатый формат строк. По сравнению с импортом данных в распакованную таблицу такой импорт может потребовать больше ресурсов ЦП.  
Если массовый импорт данных производится в кучу с включенным сжатием страниц, то при вставке данных операция массового импорта попытается применить к ним сжатие страниц.
- Сжатие не затрагивает резервное копирование и восстановление.
- Сжатие не затрагивает доставку журналов.
- Сжатие данных несовместимо с разреженными столбцами. Поэтому таблицы с

разреженными столбцами не могут быть сжаты, а разреженные столбцы не могут быть добавлены в сжатые таблицы.

- Включение сжатия может вызвать изменение планов запросов, поскольку данные будут занимать при хранении другое число страниц с другим числом строк на страницу.
- Сжатие данных поддерживается в среде Средства SQL Server Management Studio через Мастер сжатия данных.

### ***Запуск мастера сжатия данных***

- В обозревателе объектов щелкните правой кнопкой мыши таблицу, индекс или индексированное представление, укажите Хранилище и выберите команду Сжать.

### ***Наблюдение за сжатием***

Наблюдать за сжатием всего экземпляра SQL Server через счетчики Page compression attempts/sec и Pages compressed/sec объекта SQL Server, Access Methods Object.

Чтобы получить статистику сжатия отдельных секций, создайте запрос для функции динамического управления sys.dm\_db\_index\_operational\_stats.

### ***Примеры***

В некоторых из следующих примеров используются секционированные таблицы; также для этих примеров необходима база данных с файловыми группами. Создать базу данных с файловыми группами можно, выполнив следующую инструкцию.

```
CREATE DATABASE TestDatabase
ON PRIMARY
( NAME = TestDatabase,
  FILENAME = 'C:\Program Files\Microsoft SQL
Server\MSSQL10_50.MSSQLSERVER\MSSQL\Data\TestDB.mdf'),
FILEGROUP test1fg
( NAME = TestDBFile1,
  FILENAME = 'C:\Program Files\Microsoft SQL
Server\MSSQL10_50.MSSQLSERVER\MSSQL\Data\TestDBFile1.mdf'),
FILEGROUP test2fg
( NAME = TestDBFile2,
  FILENAME = 'C:\Program Files\Microsoft SQL
Server\MSSQL10_50.MSSQLSERVER\MSSQL\Data\TestDBFile2.ndf'),
FILEGROUP test3fg
( NAME = TestDBFile3,
  FILENAME = 'C:\Program Files\Microsoft SQL
Server\MSSQL10_50.MSSQLSERVER\MSSQL\Data\TestDBFile3.ndf'),
FILEGROUP test4fg
( NAME = TestDBFile4,
  FILENAME = 'C:\Program Files\Microsoft SQL
Server\MSSQL10_50.MSSQLSERVER\MSSQL\Data\TestDBFile4.ndf') ;
GO
```

Переключение на новую базу данных:

```
USE TestDatabase
GO
```

**А. Создание таблицы, в которой используется сжатие строк**

В следующем примере создается таблица, и для нее устанавливается сжатие ROW.

```
CREATE TABLE T1
(c1 int, c2 nvarchar(50) )
WITH (DATA_COMPRESSION = ROW);
GO
```

**Б. Создание таблицы, в которой используется сжатие страниц**

В следующем примере создается таблица, и для нее устанавливается сжатие PAGE.

```
CREATE TABLE T2
(c1 int, c2 nvarchar(50) )
WITH (DATA_COMPRESSION = PAGE);
GO
```

**В. Установка параметра DATA\_COMPRESSION для секционированной таблицы**

В следующем примере используется таблица TestDatabase, созданная с помощью кода, предоставленного ранее в этом разделе. Этот пример создает функцию секционирования и схему секционирования, а затем создает секционированную таблицу и указывает параметры сжатия для секций в этой таблице. В этом примере для секции 1 устанавливается сжатие ROW, а для остальных — сжатие PAGE.

Создание функции секционирования:

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

Создание схемы секционирования:

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

Создание секционированной таблицы, имеющей сжатые секции:

```
CREATE TABLE PartitionTable1
(col1 int, col2 varchar(max))
ON myRangePS1 (col1)
WITH
(
    DATA_COMPRESSION = ROW ON PARTITIONS (1),
    DATA_COMPRESSION = PAGE ON PARTITIONS (2 TO 4)
);
GO
```



### Г. Установка параметра DATA\_COMPRESSION для секционированной таблицы

В следующем примере используется база данных, которая была использована в примере В. Этот пример создает таблицу с помощью синтаксиса для несмежных секций.

```
CREATE TABLE PartitionTable2
(col1 int, col2 varchar(max))
ON myRangePS1 (col1)
WITH
(
    DATA_COMPRESSION = ROW ON PARTITIONS (1,3),
    DATA_COMPRESSION = NONE ON PARTITIONS (2,4)
);
GO
```

### Д. Изменение таблицы для изменения режима сжатия

В следующем примере изменяется сжатие для несекционированной таблицы, созданной в примере А.

```
ALTER TABLE T1
REBUILD WITH (DATA_COMPRESSION = PAGE);
GO
```

### Е. Изменение сжатия для одной секции в секционированной таблице

В следующем примере изменяется режим сжатия секционированной таблицы, созданной в примере В. Синтаксис REBUILD PARTITION = 1 вызывает перестройку только секции с номером 1.

```
ALTER TABLE PartitionTable1
REBUILD PARTITION = 1 WITH (DATA_COMPRESSION = NONE) ;
GO
```

Та же операция, использующая следующий альтернативный синтаксис, вызывает повторное построение всех секций в таблице.

```
ALTER TABLE PartitionTable1
REBUILD PARTITION = ALL
WITH (DATA_COMPRESSION = PAGE ON PARTITIONS(1) ) ;
GO
```

### Ж. Изменение режима сжатия для нескольких секций в секционированной таблице

Синтаксис REBUILD PARTITION = ... позволяет повторно построить только одну секцию. Чтобы перестроить несколько секций, необходимо либо выполнить несколько инструкций, либо выполнить следующий пример, перестраивающий все секции с использованием текущих параметров сжатия для секций, которые не указаны.

```
ALTER TABLE PartitionTable1
REBUILD PARTITION = ALL
WITH
```

```
(  
DATA_COMPRESSION = PAGE ON PARTITIONS(1),  
DATA_COMPRESSION = ROW ON PARTITIONS(2 TO 4)  
) ;  
GO
```

### 3. Изменение режима сжатия индекса

В следующем примере используется таблица, созданная в примере А, для которой создается индекс на столбце C2.

```
CREATE NONCLUSTERED INDEX IX_INDEX_1  
    ON T1 (C2)  
WITH ( DATA_COMPRESSION = ROW ) ;  
GO
```

Выполните следующий код, чтобы изменить индекс в режим сжатия страниц:

```
ALTER INDEX IX_INDEX_1  
ON T1  
REBUILD WITH ( DATA_COMPRESSION = PAGE ) ;  
GO
```

### И. Изменение режима сжатия одной секций в секционированном индексе

В следующем примере создается индекс для секционированной таблицы, в которой сжатие строк применяется во всех секциях индекса.

```
CREATE CLUSTERED INDEX IX_PartTab2Col1  
ON PartitionTable1 (Col1)  
WITH ( DATA_COMPRESSION = ROW ) ;  
GO
```

Чтобы создать индекс, использующий различные настройки сжатия для разных секций, используйте синтаксис ON PARTITIONS. В следующем примере создается индекс для секционированной таблицы, в которой применяется сжатие строк для секции 1 индекса и сжатие страниц для секций индекса со 2-ой по 4-ю.

```
CREATE CLUSTERED INDEX IX_PartTab2Col1  
ON PartitionTable1 (Col1)  
WITH (DATA_COMPRESSION = ROW ON PARTITIONS(1),  
    DATA_COMPRESSION = PAGE ON PARTITIONS (2 TO 4 ) ) ;  
GO
```

В следующем примере изменяется режим сжатия секционированного индекса.

```
ALTER INDEX IX_PartTab2Col1 ON PartitionTable1  
REBUILD PARTITION = ALL  
WITH ( DATA_COMPRESSION = PAGE ON PARTITIONS(1) ) ;  
GO
```

## К. Изменение режима сжатия для нескольких секций в секционированном индексе

Синтаксис REBUILD PARTITION = ... позволяет повторно построить только одну секцию. Чтобы перестроить несколько секций, необходимо либо выполнить несколько инструкций, либо выполнить следующий пример, перестраивающий все секции с использованием текущих параметров сжатия для секций, которые не указаны.

```
ALTER INDEX IX_PartTab2Col1 ON PartitionTable1
REBUILD PARTITION = ALL
WITH
(
  DATA_COMPRESSION = PAGE ON PARTITIONS(1),
  DATA_COMPRESSION = ROW ON PARTITIONS(2 TO 4)
);
GO
```

### Реализация сжатия строк

Этот раздел содержит описание того, как компонент Database Engine реализует сжатие строки. В этой сводке представлены основные сведения, которые помогут при планировании объема хранения.

Включение сжатия изменяет только формат физического хранения данных, связанный с типом данных, но не с синтаксисом или семантикой. Изменения приложения не требуются, если одна или более таблиц разрешены для сжатия. Новый формат хранения записи содержит следующие основные изменения.

- Он уменьшает нагрузку на метаданные, связанные с записью. Эти метаданные являются носителями информации о столбцах, их длине и смещениях. В некоторых случаях нагрузка на метаданные может быть больше, чем в старом формате хранения.
- В нем используется переменная длина для числовых типов (например, integer, decimal и float) и для типов, основанных на числовых типах (например, datetime и money).
- Он сохраняет фиксированные символьные строки, используя формат переменной длины, не сохраняя пустые символы.

#### Примечание

Значения NULL и 0 всех типов данных оптимизированы и не занимают места.

### Влияние сжатия строки на хранение

В следующей таблице описано, как сжатие строки влияет на существующие типы в SQL Server. Данная таблица не включает сохранение, которое можно осуществить, используя сжатие страницы.

| Тип данных | Изменилось ли хранение? | Описание                                      |
|------------|-------------------------|-----------------------------------------------|
| tinyint    | Нет                     | 1 байт является минимальным объемом хранения. |

|            |    |                                                                                                                                                                                                                                                                                                                                                                    |
|------------|----|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| smallint   | Да | Если значение соответствует 1 байту, то будет использован только 1 байт.                                                                                                                                                                                                                                                                                           |
| int        | Да | Использует минимально необходимое число байт. Например, если значение может храниться в 1 байте, то для его хранения будет использоваться всего 1 байт.                                                                                                                                                                                                            |
| bigint     | Да | Использует минимально необходимое число байт. Например, если значение может храниться в 1 байте, то для его хранения будет использоваться всего 1 байт.                                                                                                                                                                                                            |
| decimal    | Да | Тот же тип хранения, что и для формата vardecimal. Дополнительные сведения см. в разделе <a href="#">Хранение десятичных данных в виде значений переменной длины</a> .                                                                                                                                                                                             |
| numeric    | Да | Тот же тип хранения, что и для формата vardecimal. Дополнительные сведения см. в разделе <a href="#">Хранение десятичных данных в виде значений переменной длины</a> .                                                                                                                                                                                             |
| bit        | Да | Издержки на метаданные приводят это значение к 4 битам.                                                                                                                                                                                                                                                                                                            |
| smallmoney | Да | Использует целочисленное представление данных с помощью 4-битового целого числа. Значение валюты умножается на 10000, а полученное целочисленное значение сохраняется с удалением всех нулей после десятичной запятой. При хранении этого типа выполняется такая же оптимизация, что и для целочисленных типов.                                                    |
| money      | Да | Использует целочисленное представление данных с помощью 8-битового целого числа. Значение валюты умножается на 10000, а полученное целочисленное значение сохраняется с удалением всех нулей после десятичной запятой. Данный тип имеет больший диапазон, чем smallmoney. При хранении этого типа выполняется такая же оптимизация, что и для целочисленных типов. |
| float      | Да | Наименее значащие байты, содержащие нули, не сохраняются. В большинстве случаев для недробных значений в мантиссе применяется сжатие float.                                                                                                                                                                                                                        |

|               |     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
|---------------|-----|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| real          | Да  | Наименее значащие байты, содержащие нули, не сохраняются. В большинстве случаев для недробных значений в мантиссе применяется сжатие real.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| smalldatetime | Нет | Использует целочисленное представление данных с использованием двухбайтных значений. Дата занимает 2 байта. Время представляет собой количество дней, прошедших с 01.01.1901. Начиная с 1902 г. необходимо 2 байта. Поэтому начиная с этой даты сэкономить место нельзя. Время представляет количество минут начиная с полуночи. Значения времени, находящиеся немного позже 4 часов утра, начинают использовать второй байт. Если тип данных smalldatetime используется только для представления даты (обычно), время равно 0,0. При сжатии экономится 2 байта с помощью хранения времени в наиболее значимом байтовом формате для сжатия строки. |
| datetime      | Да  | Использует целочисленное представление данных с использованием двухбайтных значений. Целочисленное значение представляет количество дней, прошедших с 01.01.1900. Первые 2 байта могут представлять даты до 2079 г. Сжатие может сэкономить 2 байта до этой даты. Каждое целочисленное значение представляет собой 3,33 миллисекунды. Сжатие исчерпывает первые 2 байта в первые пять минут после 4:00, и становится необходимо четыре байта. Таким образом, сжатие может сэкономить только 1 байт после 4:00. Если тип данных datetime сжат, как другие целые числа, сжатие экономит 2 байта в дате.                                              |
| date          | Нет | Использует целочисленное представление данных с использованием трехбайтных значений. Это представляет дату начиная с 01.01.0001. Для современных дат сжатие строк использует все 3 байта. При этом экономии не получается.                                                                                                                                                                                                                                                                                                                                                                                                                         |
| time          | Нет | Использует целочисленное представление данных длиной от 3 до 6 байт. Имеются различные степени точности, от 0 до 9, которые могут использовать от 3 до 6 байтов. Сжатое место используется                                                                                                                                                                                                                                                                                                                                                                                                                                                         |

|                |    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
|----------------|----|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|                |    | <p>следующим образом.</p> <ul style="list-style-type: none"> <li>• Точность = 0. Байты = 3. Каждое целочисленное значение представляет собой одну секунду. Сжатие может представлять время до 18:00, используя 2 байта, потенциально экономя 1 байт.</li> <li>• Точность = 1. Байты = 3. Каждое целочисленное значение представляет собой 1/10 секунды. Сжатие использует третий байт до 2:00. В результате возникает небольшая экономия.</li> <li>• Точность = 2. Байты = 3. Подобно предыдущему случаю, экономия маловероятна.</li> <li>• Точность = 3. Байты = 4. Поскольку до 05:00 берутся первые 3 байта, возникает небольшая экономия.</li> <li>• Точность = 4. Байты = 4. Первые 3 байта берутся в первые 27 секунд. Экономии не ожидается.</li> <li>• Точность = 5, байты = 5. Пятый байт будет использован после 12:00.</li> <li>• Точность = 6 и 7, байты = 5. При этом экономии не получается.</li> <li>• Точность = 8, байты = 6. Шестой байт будет использован после 03:00.</li> </ul> <p>Хранение сжатой строки происходит без изменений. В целом при сжатии типа данных time может ожидать небольшая экономия.</p> |
| datetime2      | Да | <p>Использует целочисленное представление данных длиной от 6 до 9 байт. Первые 4 байта представляют дату. Количество байтов, берущихся для времени, будет зависеть от указанной точности времени. Это целочисленное значение обозначает количество дней, прошедших с 01.01.0001, и ограничено верхней границей 31.12.9999. При представлении даты в 2005 г. сжатие экономит 3 байта.</p> <p>Экономии при хранении времени не получается, так как оно занимает от 2 до 4 байт в зависимости от точности. Таким образом, при точности в одну секунду сжатие использует для времени 2 байта, начиная использовать второй байт после 255 секунд.</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| datetimeoffset | Да | Напоминает тип данных datetime2, за                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |

|                        |     |                                                                                                                                                                                                                                                                                                                                          |
|------------------------|-----|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|                        |     | <p>исключением того, что использует 2 байта часового пояса в формате (НН:ММ). Так же как и для типа datetime2, сжатие может сохранить 2 байта.</p> <p>Для значений часового пояса в большинстве случаев значение ММ может быть равно 0. Таким образом, сжатие может сохранить 1 байт.</p> <p>Изменений в хранении сжатой строки нет.</p> |
| char                   | Да  | <p>Конечные символы заполнения удаляются. Обратите внимание, что компонент Database Engine вставляет один и тот же символ заполнения, независимо от используемых параметров сортировки.</p>                                                                                                                                              |
| varchar                | Нет | Не влияет.                                                                                                                                                                                                                                                                                                                               |
| text                   | Нет | Не влияет.                                                                                                                                                                                                                                                                                                                               |
| nchar                  | Да  | <p>Текст сжимается при помощи алгоритма стандартной схемы сжатия Юникода (SCSU), если размер сжатого текста не превышает размера самого текста.</p>                                                                                                                                                                                      |
| nvarchar               | Да  | <p>Текст сжимается при помощи алгоритма стандартной схемы сжатия Юникода (SCSU), если размер сжатого текста не превышает размера самого текста.</p> <p><b>Примечание</b><br/>Сжатие не поддерживается для типа nvarchar(max).</p>                                                                                                        |
| ntext                  | Нет | Не влияет.                                                                                                                                                                                                                                                                                                                               |
| binary                 | Да  | Замыкающие нули удаляются.                                                                                                                                                                                                                                                                                                               |
| varbinary              | Нет | Не влияет.                                                                                                                                                                                                                                                                                                                               |
| image                  | Нет | Не влияет.                                                                                                                                                                                                                                                                                                                               |
| cursor                 | Нет | Не влияет.                                                                                                                                                                                                                                                                                                                               |
| timestamp / rowversion | Да  | <p>Использует целочисленное представление данных с использованием трехбайтных значений. Значение счетчика отметок времени поддерживается для каждой базы данных, и это значение начинается с 0. Сжатие возможно и с другими целыми значениями.</p>                                                                                       |

|                                 |     |                                           |
|---------------------------------|-----|-------------------------------------------|
| sql_variant                     | Нет | Не влияет.                                |
| uniqueidentifier                | Нет | Не влияет.                                |
| table                           | Нет | Не влияет.                                |
| xml                             | Нет | Не влияет.                                |
| Определяемые пользователем типы | Нет | Имеет внутреннее представление varbinary. |
| FILESTREAM                      | Нет | Имеет внутреннее представление varbinary. |

### Реализация сжатия страниц

Этот раздел содержит описание того, как компонент Database Engine реализует сжатие страницы. В этой сводке представлены основные сведения, которые помогут при планировании объема хранения.

#### Примечание

Подробности сжатия данных могут меняться без предварительного уведомления в пакетах обновлений или в последующих версиях.

Сжатие страницы похоже на сжатие таблиц, табличных секций, индексов и индексных секций. Последующее описание сжатия страницы для таблицы одинаково применимо для сжатия страницы для объектов всех типов. В следующих примерах сжимаются символьные строки, но при сжатии префикса и словаря применяются те же принципы, что и для других типов данных.

Сжатие конечного уровня таблиц и индексов вместе со сжатием страницы состоит из трех операций в следующем порядке:

1. сжатие строк;
2. сжатие префикса;
3. сжатие словаря.

При сжатии страницы неконечные страницы индексов сжимаются, используя только сжатие строк. Дополнительные сведения о сжатии строк см. в разделе [Реализация сжатия строк](#).

### Сжатие префикса

Для каждой сжимаемой страницы сжатие префикса состоит из следующих шагов.

1. Для каждого столбца определяется значение, которое может использоваться для уменьшения места хранения значений в каждом столбце.
2. Строка, представляющая префиксные значения для каждого столбца, создается и хранится в структуре сжатой информации (CI), следующей за заголовком страницы.
3. Повторяющиеся префиксные значения в данном столбце заменяются ссылкой на соответствующий префикс. Если значение строки совпадает с выбранным префиксным значением не полностью, то частичные совпадения все еще могут быть отражены.



Следующая иллюстрация показывает образец страницы таблицы перед сжатием префикса.

| Заголовок страницы |        |      |
|--------------------|--------|------|
|                    |        |      |
| aaabb              | aaaab  | abcd |
| aaabcc             | bbbb   | abcd |
| aaacc              | aaaacc | bbbb |

Следующая иллюстрация содержит эту же страницу после сжатия префикса. Префикс перемещается к заголовку, а значения столбца изменяются ссылками на данный префикс.

| Заголовок страницы |         |         |
|--------------------|---------|---------|
| aaabcc             | aaaacc  | abcd    |
|                    |         |         |
| 4b                 | 4b      | [пусто] |
| [пусто]            | [0bbbb] | [пусто] |
| 3ccc               | [пусто] | [0bbbb] |

В первом столбце первой строки значение 4b указывает, что первые четыре символа префикса (aaab) присутствуют для этой строки вместе с символом b. Это делает результирующим значение aaabb, которое представляет собой исходное значение.

### Сжатие словаря

После завершения сжатия префикса применяется сжатие словаря. При сжатии словаря выполняется поиск на странице повторяющихся значений и сохранение их в области CI. В отличие от сжатия префикса, сжатие словаря не ограничено одним столбцом. При сжатии словаря можно заменить повторяющиеся значения, встречающиеся в любом месте страницы. Следующая иллюстрация содержит эту же страницу после сжатия словаря.

| Заголовок страницы |         |         |
|--------------------|---------|---------|
| aaabcc             | aaaacc  | abcd    |
| 4b                 | [0bbbb] |         |
| 0                  | 0       | [пусто] |
| [пусто]            | 1       | [пусто] |
| 3ccc               | [пусто] | 1       |

Обратите внимание, что ссылки на значение 4b существуют в различных столбцах страницы.

### Когда происходит сжатие страницы

При создании новой таблицы, имеющей сжатие страницы, сжатия не происходит. Однако табличные метаданные указывают на то, что сжатие страницы должно быть использовано. Так как данные добавляются на первую страницу данных, они имеют сжатие строк. Поскольку страница не является полной, преимуществ сжатие страницы не дает. После того как страница заполнена, добавление следующей строки вызывает операцию ее сжатия. Вся страница просматривается; каждый столбец оценивается для сжатия префикса, а затем оцениваются все столбцы для сжатия словаря. Если при сжатии на странице освободилось достаточно места для дополнительной строки, то добавляется строка и в обоих случаях данные являются сжатыми по строкам и по страницам. Если пространство на странице сжатия минус пространство, указанное для структуры CI, увеличилось незначительно, сжатие страницы не используется для данной страницы. Любые будущие строки помещаются на другой странице, или, если они там не помещаются, в таблицу включается новая страница. Аналогично первой странице новая страница не является сразу сжатой постранично.

Если существующая таблица, содержащая данные, преобразуется для сжатия страниц, каждая страница перестраивается и оценивается. Перестроение всех страниц приводит к перестроению таблицы, индекса или секции.

### Реализация сжатия Юникода

В SQL Server используется реализация алгоритма стандартной схемы сжатия Юникода (SCSU), которая сохраняет данные в сжатых объектах строк или страниц. Для этих объектов сжатие Юникода для столбцов типов `nchar(n)` и `nvarchar(n)` выполняется автоматически. Компонент Database Engine хранит данные Юникода в виде 2 байтовых символов, независимо от локали. Такая кодировка называется UCS-2. Для некоторых локалей реализация сжатия по SCSU в SQL Server может сэкономить до 50% места в хранилище.

### Поддерживаемые типы данных

Сжатие Юникода поддерживает типы данных фиксированной длины `nchar(n)` и `nvarchar(n)`. Внестрочные значения данных и значения, хранящиеся в

столбцах `nvarchar(max)`, не сжимаются.

#### Примечание

Для данных `nvarchar(max)` не поддерживается сжатие Юникода, даже если они хранятся в строке. Но этот тип данных дает преимущество при сжатии страниц.

### Обновление с предыдущих версий SQL Server

При обновлении базы данных SQL Server до SQL Server 2008 R2 все изменения, связанные со сжатием Юникода, не оказывают влияния на объекты базы данных, сжатые или распакованные. После обновления базы данных ситуация с объектами выглядит следующим образом.

- Если объект не сжат, то никаких изменений не происходит и объект продолжает работать как раньше.
- Сжатые объекты строк и страниц продолжают работу как раньше. Распакованные данные остаются в распакованной форме до тех пор, пока их значение не будет обновлено.
- Для новых строк, вставляемых в таблицу, сжатую на уровне строк или страниц, производится сжатие Юникода.

#### Примечание

Для использования всех преимуществ сжатия Юникода объект необходимо перестроить с использованием сжатия страниц или строк.

### Влияние сжатия Юникода на хранилище данных

При создании или перестройке индекса, а также при изменении значения в таблице, сжатой на уровне строк или страниц, соответствующий индекс или значение сохраняется в сжатом виде только в том случае, если его размер в сжатом виде будет меньше текущего. Сжатие Юникода позволяет предотвратить разрастание размера строк в таблице или индексе.

Объем места в хранилище, сэкономленного благодаря сжатию, зависит от характеристик сжимаемых данных и локали данных. Следующая таблица содержит данные по экономии, достижимой для некоторых локалей.

| Локаль          | Процент сжатия |
|-----------------|----------------|
| Английский язык | 50%            |
| Немецкий язык   | 50%            |
| Хинди           | 50%            |
| Турецкий язык   | 48%            |
| Вьетнамский     | 39%            |
| Японский язык   | 15%            |

## Просмотр таблиц

После того как таблица создана в базе данных, можно получать сведения об ее свойствах: именах и типах данных столбцов, природе индексов, и, что еще более важно, можно просматривать данные, содержащиеся в этой таблице.

Кроме того, можно определить зависимости таблицы, чтобы определить, какие объекты (представления, хранимые процедуры, триггеры) от нее зависят. Если в таблицу вносятся какие-либо изменения, это может отразиться на зависящих от нее объектах.

## sp\_help (Transact-SQL)

Возвращает сведения об объекте базы данных (любом объекте, присутствующем в представлении совместимости **sys.sysobjects**), обычном или определяемом пользователем типе данных.

[Синтаксические обозначения в Transact-SQL](#)

### Синтаксис

```
sp_help [ [ @objname = ] 'name' ]
```

### Аргументы

[ @objname=] 'name'

Имя любого объекта в **sysobjects** или любого пользовательского типа данных в таблице **systypes**. Аргумент name имеет тип **nvarchar(776)** и значение по умолчанию NULL. Имена баз данных неприемлемы.

### Значения кодов возврата

0 (успешное завершение) или 1 (ошибка)

### Результирующие наборы

Возвращаемые результирующие наборы зависят от того, указан ли аргумент name, когда он указан, и какой объект базы данных имеется в виду.

1. Если процедура **sp\_help** выполняется без аргументов, возвращаются общие сведения об объектах всех типов, существующих в текущей базе данных.

| Имя столбца | Тип данных    | Описание         |
|-------------|---------------|------------------|
| Название    | nvarchar(128) | Имя объекта      |
| Owner       | nvarchar(128) | Владелец объекта |
| Object_type | nvarchar(31)  | Тип объекта      |

2. Если аргумент `name` является типом данных SQL Server или пользовательским типом данных, то процедура **sp\_help** возвращает этот результирующий набор.

| Имя столбца                 | Тип данных    | Описание                                                                                                |
|-----------------------------|---------------|---------------------------------------------------------------------------------------------------------|
| <b>Type_name</b>            | nvarchar(128) | Имя типа данных.                                                                                        |
| <b>Storage_type</b>         | nvarchar(128) | Имя типа данных SQL Server.                                                                             |
| <b>Length</b>               | smallint      | Физическая длина типа данных (в байтах).                                                                |
| <b>Prec</b>                 | int           | Точность (общее количество знаков).                                                                     |
| <b>Масштаб</b>              | int           | Количество знаков справа от десятичной запятой.                                                         |
| <b>Да</b>                   | varchar(35)   | Указывает, допускаются ли значения NULL. Возможны значения Yes (да) или No (нет).                       |
| <b>Default_name</b>         | nvarchar(128) | Имя значения по умолчанию, привязанного к этому типу.<br>NULL = Нет привязанного значения по умолчанию. |
| <b>Rule_name</b>            | nvarchar(128) | Имя правила, привязанного к этому типу.<br>NULL = Нет привязанного правила по умолчанию.                |
| <b>Параметры сортировки</b> | sysname       | Параметры сортировки для типа данных. Имеет значение NULL для несимвольных типов данных.                |

3. Если аргумент `name` соответствует любому объекту базы данных, кроме типа данных, процедура **sp\_help** возвращает этот результирующий набор, а также дополнительные результирующие наборы, в зависимости от указанного типа объектов.

| Имя столбца             | Тип данных    | Описание              |
|-------------------------|---------------|-----------------------|
| <b>Название</b>         | nvarchar(128) | Имя таблицы           |
| <b>Owner</b>            | nvarchar(128) | Владелец таблицы      |
| <b>Тип</b>              | nvarchar(31)  | Тип таблицы           |
| <b>Created_datetime</b> | datetime      | Дата создания таблицы |

В зависимости от указанного объекта базы данных, процедура **sp\_help** возвращает дополнительные результирующие наборы.

Если аргумент name соответствует системной таблице, пользовательской таблице или представлению, процедура **sp\_help** возвращает следующие результирующие наборы. Однако результирующий набор, описывающий место расположения файла данных внутри файловой группы, для представления не возвращается.

- Дополнительный результирующий набор, возвращаемый для объектов столбца:

| Имя столбца                 | Тип данных    | Описание                                                                                                                                                                                 |
|-----------------------------|---------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Column_name</b>          | nvarchar(128) | Имя столбца.                                                                                                                                                                             |
| <b>Тип</b>                  | nvarchar(128) | Тип данных столбца.                                                                                                                                                                      |
| <b>Computed</b>             | varchar(35)   | Указывает, являются ли значения в столбце вычисляемыми: Yes или No.                                                                                                                      |
|                             |               | Длина столбца в байтах.                                                                                                                                                                  |
| <b>Length</b>               | int           | <b>Примечание</b><br>Если для столбца задан тип больших значений ( <b>varchar(max)</b> , <b>nvarchar(max)</b> , <b>varbinary(max)</b> или <b>xml</b> ), то будет отображено значение -1. |
| <b>Prec</b>                 | char(5)       | Точность столбца.                                                                                                                                                                        |
| <b>Масштаб</b>              | char(5)       | Масштаб столбца.                                                                                                                                                                         |
| <b>Да</b>                   | varchar(35)   | Указывает, допустимы ли для столбца значения NULL: Yes или No.                                                                                                                           |
| <b>TrimTrailingBlanks</b>   | varchar(35)   | Указывает, усекают ли завершающие пробелы или нет. Возвращает значение Yes или No.                                                                                                       |
| <b>FixedLenNullInSource</b> | varchar(35)   | Только для обратной совместимости.                                                                                                                                                       |
| <b>Параметры сортировки</b> | sysname       | Параметры сортировки столбца. Имеет значение NULL для несимвольных типов данных.                                                                                                         |

- Дополнительный результирующий набор, возвращаемый для столбцов идентификаторов:

| Имя столбца                | Тип данных    | Описание                                                                                                                                                              |
|----------------------------|---------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Identity</b>            | nvarchar(128) | Имя столбца, чей тип данных объявлен удостоверением.                                                                                                                  |
| <b>Seed</b>                | numeric       | Стартовое значение для столбца идентификаторов.                                                                                                                       |
| <b>Increment</b>           | numeric       | Шаг прироста, который следует использовать для значений в этом столбце.                                                                                               |
| <b>Not For Replication</b> | int           | Свойство IDENTITY не обеспечивается принудительно, когда имя входа репликации, например <b>sqlrepl</b> вставляет данные в таблицу:<br><br>1 = True;<br><br>0 = False. |

- Дополнительный результирующий набор, возвращаемый для столбцов:

| Имя столбца       | Тип данных | Описание                                            |
|-------------------|------------|-----------------------------------------------------|
| <b>RowGuidCol</b> | sysname    | Имя столбца глобального уникального идентификатора. |

- Дополнительный результирующий набор, возвращаемый для файловых групп:

| Имя столбца                      | Тип данных    | Описание                                                                                         |
|----------------------------------|---------------|--------------------------------------------------------------------------------------------------|
| <b>Data_located_on_filegroup</b> | nvarchar(128) | Файловая группа, в которой расположены данные: главная, вторичная или группа журнала транзакций. |

- Дополнительный результирующий набор, возвращаемый для индексов:

| Имя столбца              | Тип данных     | Описание                                           |
|--------------------------|----------------|----------------------------------------------------|
| <b>index_name</b>        | sysname        | Имя индекса.                                       |
| <b>Index_description</b> | varchar(210)   | Описание индекса.                                  |
| <b>index_keys</b>        | nvarchar(2078) | Имена столбцов, на основе которых построен индекс. |

- Дополнительный результирующий набор, возвращаемый для ограничений:

| Имя столбца                   | Тип данных     | Описание                                                                                                                                                                        |
|-------------------------------|----------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>constraint_type</b>        | nvarchar(146)  | Тип ограничения.                                                                                                                                                                |
| <b>constraint_name</b>        | nvarchar(128)  | Имя ограничения.                                                                                                                                                                |
| <b>delete_action</b>          | nvarchar(9)    | Указывает, является ли действие DELETE: No Action, CASCADE или N/A.<br>Применимо только для ограничений FOREIGN KEY.                                                            |
| <b>update_action</b>          | nvarchar(9)    | Указывает, является ли действие UPDATE: No Action, Cascade или N/A. SET_NULL и SET_DEFAULT отображаются как «Нет действия».<br>Применимо только для ограничений FOREIGN KEY.    |
| <b>status_enabled</b>         | varchar(8)     | Указывает, включено ли ограничение: Enabled, Disabled или N/A. SET_NULL и SET_DEFAULT отображаются как «Нет действия».<br>Применимо только для ограничений CHECK и FOREIGN KEY. |
| <b>status_for_replication</b> | varchar(19)    | Указывает, предназначено ли ограничение для репликации.<br>Применимо только для ограничений CHECK и FOREIGN KEY.                                                                |
| <b>constraint_keys</b>        | nvarchar(2078) | Имена столбцов, составляющих ограничение, или, в случае со значениями по умолчанию и правилами, текст, определяющий значение по умолчанию или правило.                          |

- Дополнительный результирующий набор, возвращаемый для ссылочных объектов:

| Имя столбца                   | Тип данных    | Описание                                                            |
|-------------------------------|---------------|---------------------------------------------------------------------|
| <b>Table is referenced by</b> | nvarchar(516) | Указывает другие объекты базы данных, которые ссылаются на таблицу. |



- Дополнительный результирующий набор, возвращаемый для хранимых процедур, функций или расширенных хранимых процедур.

| Имя столбца    | Тип данных    | Описание                                           |
|----------------|---------------|----------------------------------------------------|
| Parameter_name | nvarchar(128) | Имя аргумента хранимой процедуры.                  |
| Тип            | nvarchar(128) | Тип данных аргумента хранимой процедуры.           |
| Length         | smallint      | Максимальная физическая длина хранилища, в байтах. |
| Prec           | int           | Точность или общее количество знаков.              |
| Масштаб        | int           | Количество цифр справа от десятичного знака.       |
| Param_order    | smallint      | Порядок аргумента.                                 |

## Замечания

Процедура **sp\_help** осуществляет поиск объекта только в текущей базе данных.

Если аргумент name не указан, процедура **sp\_help** перечисляет имена объектов, владельцев и типы объектов для всех объектов в текущей базе данных. Процедура **sp\_helptrigger** предоставляет информацию о триггерах.

Процедура **sp\_help** предоставляет доступ только к упорядочиваемым столбцам индекса; поэтому она не предоставляет доступа к XML-индексам или пространственным индексам.

## Разрешения

Необходимо членство в роли **public**. Пользователь должен иметь по меньшей мере одно разрешение для objname. Чтобы просмотреть ключи, значения по умолчанию или правила ограничения для столбца, необходимо обладать разрешением VIEW DEFINITION для этой таблицы.

## Примеры

### А. Возвращение сведений обо всех объектах

В нижеследующем примере приводится информация о каждом объекте в базе данных master.

```
USE master;
GO
EXEC sp_help;
GO
```

## Б. Возвращение сведений об отдельном объекте

В нижеследующем примере отображаются сведения о столбце Person в таблице Person.

```
USE AdventureWorks2008R2;
GO
EXEC sp_help 'Person.Person';
GO
```

## SELECT (Transact-SQL)

Возвращает строки из базы данных и позволяет делать выборку одной или нескольких строк или столбцов из одной или нескольких таблиц в SQL Server 2008 R2. Полный синтаксис инструкции SELECT сложен, однако основные предложения можно вкратце описать следующим образом:

```
[ WITH <common_table_expression> ]
SELECT select_list [ INTO new_table ]
[ FROM table_source ] [ WHERE search_condition ]
[ GROUP BY group_by_expression ]
[ HAVING search_condition ]
[ ORDER BY order_expression [ ASC | DESC ] ]
```

Операторы UNION, EXCEPT и INTERSECT можно использовать между запросами, чтобы сравнить их результаты или объединить в один результирующий набор.

### *Синтаксис*

<SELECT statement> ::=

```
    [WITH <common_table_expression> [,...n]]
    <query_expression>
    [ ORDER BY { order_by_expression | column_position [ ASC | DESC ] }
    [ ,...n ] ]
    [ COMPUTE
    { { AVG | COUNT | MAX | MIN | SUM } (expression) } [ ,...n ]
    [ BY expression [ ,...n ] ]
    ]
    [ <FOR Clause> ]
    [ OPTION ( <query_hint> [ ,...n ] ) ]
```

<query\_expression> ::=

```
    { <query_specification> | ( <query_expression> ) }
    [ { UNION [ ALL ] | EXCEPT | INTERSECT }
      <query_specification> | ( <query_expression> ) [...n ] ]
```

```
<query_specification> ::=  
SELECT [ ALL | DISTINCT ]  
    [ TOP (expression) [ PERCENT ] [ WITH TIES ] ]  
  
    < select_list >  
    [ INTO new_table ]  
    [ FROM { <table_source> } [ ,...n ] ]  
    [ WHERE <search_condition> ]  
    [ <GROUP BY> ]  
    [ HAVING < search_condition > ]
```

### ***Замечания***

Учитывая сложность инструкции SELECT, элементы ее синтаксиса и аргументы подробно представлены в предложении:

[WITH обобщенное табличное выражение](#)   [UNION](#)

[Предложение SELECT](#)

[EXCEPT и INTERSECT](#)

[Предложение INTO](#)

[ORDER BY](#)

[FROM](#)

[COMPUTE](#)

[WHERE](#)

[Предложение FOR](#)

[GROUP BY](#)

[Предложение OPTION](#)

[HAVING](#)

Порядок предложений в инструкции SELECT имеет значение. Любое из необязательных предложений может быть опущено; но если необязательные предложения используются, они должны следовать в определенном порядке.

Инструкции SELECT разрешено использовать в определяемых пользователем функциях только в том случае, если списки выбора этих инструкций содержат выражения, которые присваивают значения переменным, локальным для функций.

Четырехкомпонентное имя, составленное с помощью функции OPENDATASOURCE как части server-name, может быть использовано в качестве исходной таблицы в любом месте инструкции SELECT, где может появляться имя таблицы.

Для инструкций SELECT, которые задействуют удаленные таблицы, существуют некоторые ограничения на синтаксис. Дополнительные сведения см. в разделе [Рекомендации по использованию распределенных запросов](#).

### ***Логический порядок обработки инструкции SELECT***

Следующие действия демонстрируют логический порядок обработки, или порядок привязки, инструкции SELECT. Согласно этому порядку предложения последующих стадий получают доступ к объектам, определенным на предыдущей стадии. Например,

если обработчик запросов может привязываться (обращаться) к таблицам или представлениям, определенным в предложении FROM, то эти объекты и их столбцы становятся доступны для всех последующих стадий. С другой стороны, поскольку стадия предложения SELECT — 8, то любые псевдонимы столбцов или производные столбцы, определенные в этом предложении, не могут упоминаться в предыдущих предложениях. Однако они могут упоминаться в последующих предложениях, например в предложении ORDER BY. Обратите внимание, что фактическое физическое выполнение предложения определяется обработчиком запросов и фактический порядок выполнения может отличаться от порядка в данном списке.

1. FROM
2. ON
3. JOIN
4. WHERE
5. GROUP BY
6. WITH CUBE или WITH ROLLUP
7. HAVING
8. SELECT
9. DISTINCT
10. ORDER BY
11. TOP

### **Разрешения**

Для выборки данных требуется разрешение SELECT на таблицу или представление, которое может быть унаследовано из области более высокого уровня, например разрешение SELECT на схему или разрешение CONTROL на таблицу. Может также потребоваться членство в предопределенной роли базы данных db\_datareader или db\_owner либо в предопределенной роли сервера sysadmin. Для создания новой таблицы инструкцией SELECT INTO также необходимы разрешения CREATETABLE и ALTERSCHEMA на схему, которой принадлежит новая таблица.

### **Выражение SELECT (Transact-SQL)**

Указывает столбцы, возвращаемые запросом.

#### **Синтаксис**

```
SELECT [ ALL | DISTINCT ]  
[ TOP ( expression ) [ PERCENT ] [ WITH TIES ] ]  
<select_list>  
<select_list> ::=  
    {  
        *  
        | { table_name | view_name | table_alias }.*
```

```

| {
  [ { table_name | view_name | table_alias }. ]
    { column_name | $IDENTITY | $ROWGUID }
  | udt_column_name [ { . | :: } { { property_name | field_name }
    | method_name (argument [ ,...n ] ) } ]
  | expression
  [ [ AS ] column_alias ]
}
| column_alias = expression
} [ ,...n ]

```

## Аргументы

### ALL

Указывает, что в результирующем наборе могут появиться повторяющиеся строки. Аргумент ALL используется по умолчанию.

### DISTINCT

Указывает, что в результирующем наборе могут появиться только уникальные строки. Значения NULL считаются равными для ключевого слова DISTINCT.

TOP ( expression ) [ PERCENT ] [ WITH TIES ]

Указывает, что только заданный первый набор или процент строк будет возвращен из результирующего набора запроса. Выражение expression может быть либо числом, либо процентом строк.

В целях обратной совместимости использование TOP expression без скобок в инструкциях SELECT поддерживается, но не рекомендуется. Дополнительные сведения см. в разделе [TOP \(Transact-SQL\)](#).

< select\_list >

Столбцы, выбираемые в результирующий набор. Список выбора представляет собой серию выражений, отделяемых запятыми. Максимальное число выражений, которое можно задать в списке выбора — 4 096.

\*

Указывает на то, что все столбцы из всех таблиц и представлений в предложении FROM должны быть возвращены. Столбцы возвращаются таблицей или представлением, как указано в предложении FROM, и в порядке, в котором они находятся в таблице или представлении.

table\_name | view\_name | table\_alias.\*

Ограничивает область \* до указанной таблицы или представления.

column\_name

Имя возвращаемого столбца. Указывайте квалификатор для аргумента column\_name во избежание неоднозначных ссылок, которые могут возникнуть, если в двух таблицах из предложения FROM содержатся столбцы с

повторяющимися именами. Например, таблицы SalesOrderHeader и SalesOrderDetail в базе данных AdventureWorks2008R2 содержат столбцы с именем ModifiedDate. Если в запросе соединяются две таблицы, то данные о дате изменения из таблицы SalesOrderDetail могут быть заданы в списке выбора как SalesOrderDetail.ModifiedDate.

expression

Является константой, функцией, любым сочетанием имен столбцов, констант и функций, соединенных оператором (операторами) или вложенным запросом.

\$IDENTITY

Возвращает столбец идентификатора. Дополнительные сведения см. в разделах [IDENTITY \(свойство\) \(Transact-SQL\)](#), [ALTER TABLE \(Transact-SQL\)](#) и [Инструкция CREATE TABLE \(Transact-SQL\)](#).

Если более чем одна таблица из предложения FROM содержит столбец со свойством IDENTITY, \$IDENTITY должно быть задано с определенным именем таблицы, например T1.\$IDENTITY.

\$ROWGUID

Возвращает столбец с идентификаторами GUID строки.

Если более чем одна таблица из предложения FROM содержит столбец со свойством ROWGUIDCOL, \$ROWGUIDCOL должно быть задано с определенным именем таблицы, например T1.\$IDENTITY.

udt\_column\_name

Имя возвращаемого определяемого пользователем типа данных CLR столбца.

### Примечание

Среда SQL Server Management Studio возвращает значения определяемого пользователем типа в двоичном представлении. Чтобы вернуть значения определяемого пользователем типа в виде строки или в формате XML, используйте [CAST](#) или [CONVERT](#).

{ . | :: }

Указывает метод, свойство или поле определяемого пользователем типа данных CLR. Используйте . для метода экземпляра (нестатического), свойства или поля. Используйте :: для статического метода, свойства или поля. Для обращения к методу, свойству или полю определяемого пользователем типа CLR необходимо разрешение EXECUTE для этого типа.

property\_name

Общее свойство udt\_column\_name.

field\_name

Общий член данных `udt_column_name`.

`method_name`

Общий метод `udt_column_name`, который принимает один или более аргументов. Метод `method_name` не может быть методом-мутатором.

В следующем примере выбираются значения столбца `Location`, определенного типом `point`, из таблицы `Cities` путем обращения к методу типа, названного `Distance`:

```
CREATE TABLE Cities (  
    Name varchar(20),  
    State varchar(20),  
    Location point );  
GO  
DECLARE @p point (32, 23), @distance float  
GO  
SELECT Location.Distance (@p)  
FROM Cities;
```

`column_alias`

Альтернативное имя, которым можно заменить имя столбца в результирующем наборе запроса. Например, для столбца «quantity» может быть указан псевдоним «Quantity», «Quantity to Date» или «Qty».

Кроме того, псевдонимы используются для указания имен для результатов выражений, например:

```
USE AdventureWorks2008R2;  
  
GO  
  
SELECT AVG(UnitPrice) AS 'Average Price'  
  
FROM Sales.SalesOrderDetail;
```

Псевдоним `column_alias` может быть использован в предложении `ORDER BY`. Однако он не может быть использован в предложениях `WHERE`, `GROUP BY` или `HAVING`. Если выражение запроса является частью инструкции `DECLARE CURSOR`, `column_alias` не может быть использован в предложении `FOR UPDATE`.

### Замечания

Длина возвращаемых данных для столбцов типа `text` или `ntext`, включенных в список выбора, устанавливается в минимальное значение из следующих: реальный размер столбца типа `text`, значение настройки `TEXTSIZE` сеанса по умолчанию или жестко запрограммированное ограничение. Чтобы изменить длину возвращаемого текста для сеанса, используйте инструкцию `SET`. По умолчанию ограничение на длину возвращаемых при помощи инструкции `SELECT` текстовых данных равно 4 000 байт.

Компонент SQL Server Database Engine вызывает исключение номер 511 и осуществляет откат транзакции до момента выполнения текущей инструкции, если наблюдается одно из следующих явлений.

- Следствием инструкции SELECT является строка результата или строка промежуточной таблицы, превышающая 8 060 байт.
- Инструкции DELETE, INSERT или UPDATE производят действия со строкой, превышающей 8 060 байт.

Ошибка возникает в случае, если не указано имя столбца, созданного при помощи инструкции SELECT INTO или CREATE VIEW.

### *Примеры использования инструкции SELECT (Transact-SQL)*

#### **А. Использование инструкции SELECT для получения строк и столбцов**

В следующем примере приведены три примера кода. В ходе выполнения первого примера кода возвращаются все строки (предложение WHERE не указано), а также все столбцы (используется звездочка, \*) таблицы Product базы данных База данных AdventureWorks2008R2.

```
USE AdventureWorks2008R2;
GO
SELECT *
FROM Production.Product
ORDER BY Name ASC;
-- Alternate way.
USE AdventureWorks2008R2;
GO
SELECT p.*
FROM Production.Product AS p
ORDER BY Name ASC;
GO
```

В ходе выполнения данного примера кода происходит выдача всех строк (предложение WHERE не задано) и подмножества столбцов (Name, ProductNumber, ListPrice) таблицы Product базы данных База данных AdventureWorks2008R2. Дополнительно выведено название столбца.

```
USE AdventureWorks2008R2;
GO
SELECT Name, ProductNumber, ListPrice AS Price
FROM Production.Product
ORDER BY Name ASC;
GO
```

В ходе выполнения данного примера кода происходит выдача всех строк таблицы Product, для которых линейки продуктов начинаются символом R и для которых длительность изготовления не превышает 4 дней.

```
USE AdventureWorks2008R2;
GO
SELECT Name, ProductNumber, ListPrice AS Price
FROM Production.Product
```



```
WHERE ProductLine = 'R'  
AND DaysToManufacture < 4  
ORDER BY Name ASC;  
GO
```

## Б. Использование инструкции SELECT с заголовками столбцов и вычислениями

В ходе выполнения следующего примера возвращаются все строки таблицы Product. В результате выполнения первого примера выдаются все объемы продаж и скидки по всем продуктам. Во втором примере вычисляется годовой доход от продажи каждого вида продукции.

```
USE AdventureWorks2008R2;  
GO  
SELECT p.Name AS ProductName,  
NonDiscountSales = (OrderQty * UnitPrice),  
Discounts = ((OrderQty * UnitPrice) * UnitPriceDiscount)  
FROM Production.Product AS p  
INNER JOIN Sales.SalesOrderDetail AS sod  
ON p.ProductID = sod.ProductID  
ORDER BY ProductName DESC;  
GO
```

Данный запрос вычисляет доход от продажи по каждому виду продукции для каждого заказа.

```
USE AdventureWorks2008R2;  
GO  
SELECT 'Total income is', ((OrderQty * UnitPrice) * (1.0 - UnitPriceDiscount)),  
' for ',  
p.Name AS ProductName  
FROM Production.Product AS p  
INNER JOIN Sales.SalesOrderDetail AS sod  
ON p.ProductID = sod.ProductID  
ORDER BY ProductName ASC;  
GO
```

## В. Совместное использование DISTINCT и SELECT

В приведенном ниже примере для предотвращения получения повторяющихся заголовков используется оператор DISTINCT.

```
USE AdventureWorks2008R2;  
GO  
SELECT DISTINCT JobTitle  
FROM HumanResources.Employee  
ORDER BY JobTitle;  
GO
```

## Г. Создание таблиц с помощью инструкции SELECT INTO

В следующем примере в базе данных tempdb создается временная таблица #Bicycles.

```
USE tempdb;  
GO  
IF OBJECT_ID (N'#Bicycles',N'U') IS NOT NULL  
DROP TABLE #Bicycles;
```

```
GO
SELECT *
INTO #Bicycles
FROM AdventureWorks2008R2.Production.Product
WHERE ProductNumber LIKE 'BK%';
GO
```

В данном примере создается постоянная таблица NewProducts.

```
USE AdventureWorks2008R2;
GO
IF OBJECT_ID ('dbo.NewProducts', 'U') IS NOT NULL
    DROP TABLE dbo.NewProducts;
GO
ALTER DATABASE AdventureWorks2008R2 SET RECOVERY BULK_LOGGED;
GO

SELECT * INTO dbo.NewProducts
FROM Production.Product
WHERE ListPrice > $25
AND ListPrice < $100;
GO
ALTER DATABASE AdventureWorks2008R2 SET RECOVERY FULL;
GO
```

#### Д. Использование связанных вложенных запросов

В следующем примере представлены семантически эквивалентные запросы, а также показаны различия в использовании ключевых слов EXISTS и IN. В обоих примерах приведены допустимые вложенные запросы, извлекающие по одному экземпляру продукции каждого наименования, для которых модель продукта — «long sleeve logo jersey» (кофта с длинными рукавами, с эмблемой), а значения столбцов ProductModelID таблиц Product и ProductModel совпадают.

```
USE AdventureWorks2008R2;
GO
SELECT DISTINCT Name
FROM Production.Product AS p
WHERE EXISTS
    (SELECT *
     FROM Production.ProductModel AS pm
     WHERE p.ProductModelID = pm.ProductModelID
     AND pm.Name LIKE 'Long-Sleeve Logo Jersey%');
GO

-- OR

USE AdventureWorks2008R2;
GO
SELECT DISTINCT Name
FROM Production.Product
WHERE ProductModelID IN
    (SELECT ProductModelID
     FROM Production.ProductModel
     WHERE Name LIKE 'Long-Sleeve Logo Jersey%');
GO
```

В следующем примере в коррелированном или повторяющемся вложенном запросе

используется кодовое слово IN. Это запрос, зависящий от результатов выполнения другого запроса. Запрос повторно выполняется для каждой строки, выбранной с помощью другого запроса. Данный запрос получает имена и фамилии сотрудников, для которых значение премии в таблице SalesPerson составляет 5000.00, а соответствующие им идентификационные номера в таблицах Employee и SalesPerson совпадают.

```
USE AdventureWorks2008R2;
GO
SELECT DISTINCT p.LastName, p.FirstName
FROM Person.Person AS p
JOIN HumanResources.Employee AS e
  ON e.BusinessEntityID = p.BusinessEntityID WHERE 5000.00 IN
  (SELECT Bonus
   FROM Sales.SalesPerson AS sp
   WHERE e.BusinessEntityID = sp.BusinessEntityID);
GO
```

Предыдущий вложенный запрос данной инструкции не может быть выполнен независимо от внешнего запроса. Требуется значение параметра Employee.BusinessEntityID, однако в процессе обработки строк Employee компонентом SQL Server Database Engine указанное значение меняется.

Коррелированный вложенный запрос также может использоваться в предложении HAVING внешнего запроса. В данном примере осуществляется поиск моделей продуктов, для которых максимальная цена в каталоге в два раза превышает среднюю цену по нему.

```
USE AdventureWorks2008R2;
GO
SELECT p1.ProductModelID
FROM Production.Product AS p1
GROUP BY p1.ProductModelID
HAVING MAX(p1.ListPrice) >= ALL
  (SELECT AVG(p2.ListPrice)
   FROM Production.Product AS p2
   WHERE p1.ProductModelID = p2.ProductModelID);
GO
```

В данном примере с помощью двух коррелированных запросов осуществляется поиск сотрудников, продавших определенную продукцию.

```
USE AdventureWorks2008R2;
GO
SELECT DISTINCT pp.LastName, pp.FirstName
FROM Person.Person pp JOIN HumanResources.Employee e
  ON e.BusinessEntityID = pp.BusinessEntityID WHERE pp.BusinessEntityID IN
  (SELECT SalesPersonID
   FROM Sales.SalesOrderHeader
   WHERE SalesOrderID IN
    (SELECT SalesOrderID
     FROM Sales.SalesOrderDetail
     WHERE ProductID IN
      (SELECT ProductID
       FROM Production.Product p
       WHERE ProductNumber = 'BK-M68B-42')));
GO
```

## Е. Использование GROUP BY

В следующем примере находится общий объем продаж для каждого заказа в базе данных.

```
USE AdventureWorks2008R2;
GO
SELECT SalesOrderID, SUM(LineTotal) AS SubTotal
FROM Sales.SalesOrderDetail
GROUP BY SalesOrderID
ORDER BY SalesOrderID;
GO
```

Так как в запросе используется предложение GROUP BY, то для каждого заказа выводится только одна строка, содержащая общий объем продаж.

## Ж. Использование GROUP BY с несколькими группами

В данном примере вычисляются средние цены и объемы продаж за последний год, сгруппированные по коду продукта и идентификатору специального предложения.

```
SE AdventureWorks2008R2;
GO
SELECT ProductID, SpecialOfferID, AVG(UnitPrice) AS 'Average Price',
       SUM(LineTotal) AS SubTotal
FROM Sales.SalesOrderDetail
GROUP BY ProductID, SpecialOfferID
ORDER BY ProductID;
GO
```

## З. Использование GROUP BY и WHERE

В следующем примере после извлечения строк, содержащих цены каталога, превышающие \$1000, происходит их разделение на группы.

```
USE AdventureWorks2008R2;
GO
SELECT ProductModelID, AVG(ListPrice) AS 'Average List Price'
FROM Production.Product
WHERE ListPrice > $1000
GROUP BY ProductModelID
ORDER BY ProductModelID;
GO
```

## И. Использование GROUP BY в выражении

В следующем примере производится группировка с помощью выражения. Группировку можно производить только с помощью выражения, не содержащего агрегатных функций.

```
USE AdventureWorks2008R2;
GO
SELECT AVG(OrderQty) AS 'Average Quantity',
       NonDiscountSales = (OrderQty * UnitPrice)
FROM Sales.SalesOrderDetail
GROUP BY (OrderQty * UnitPrice)
ORDER BY (OrderQty * UnitPrice) DESC;
GO
```

**К. Использование предложения GROUP BY вместе с предложением ORDER BY**

В следующем примере для каждого типа продуктов вычисляется средняя цена, а также осуществляется сортировка полученных результатов по возрастанию.

```
USE AdventureWorks2008R2;
GO
SELECT ProductID, AVG(UnitPrice) AS 'Average Price'
FROM Sales.SalesOrderDetail
WHERE OrderQty > 10
GROUP BY ProductID
ORDER BY AVG(UnitPrice);
GO
```

**Л. Использование предложения HAVING**

В первом из приведенных ниже примеров показывается использование предложения HAVING с агрегатной функцией. В нем производится группировка строк таблицы SalesOrderDetail по коду продукта, а также удаляются строки, соответствующие продуктам, для которых средний объем заказа не превышает пяти. Во втором примере показывается использование предложения HAVING без агрегатной функции.

```
USE AdventureWorks2008R2;
GO
SELECT ProductID
FROM Sales.SalesOrderDetail
GROUP BY ProductID
HAVING AVG(OrderQty) > 5
ORDER BY ProductID;
GO
```

В данном запросе внутри предложения HAVING используется предложение LIKE.

```
USE AdventureWorks2008R2 ;
GO
SELECT SalesOrderID, CarrierTrackingNumber
FROM Sales.SalesOrderDetail
GROUP BY SalesOrderID, CarrierTrackingNumber
HAVING CarrierTrackingNumber LIKE '4BD%'
ORDER BY SalesOrderID ;
GO
```

**М. Использование предложения HAVING с предложением GROUP BY**

В следующем примере показано использование предложений GROUP BY, HAVING, WHERE и ORDER BY в одной инструкции SELECT. В результате его выполнения в группах и сводных значениях не учитываются строки, соответствующие продуктам с ценами выше \$25 и средним объемом заказов ниже 5. Также осуществляется сортировка результатов по ProductID.

```
USE AdventureWorks2008R2;
GO
SELECT ProductID
FROM Sales.SalesOrderDetail
WHERE UnitPrice < 25.00
GROUP BY ProductID
HAVING AVG(OrderQty) > 5
```

```
ORDER BY ProductID;  
GO
```

## Н. Использование предложения HAVING с функциями SUM и AVG

В следующем примере производится группировка строк таблицы SalesOrderDetail по коду продукта, а затем выводятся только те группы, для которых общий объем продаж составляет более \$1000000.00, а средний объем заказа не превышает 3.

```
USE AdventureWorks2008R2;  
GO  
SELECT ProductID, AVG(OrderQty) AS AverageQuantity, SUM(LineTotal) AS Total  
FROM Sales.SalesOrderDetail  
GROUP BY ProductID  
HAVING SUM(LineTotal) > $1000000.00  
AND AVG(OrderQty) < 3;  
GO
```

Чтобы вывести список продуктов, для которых общий объем продаж составляет более \$2000000.00, необходимо использовать следующий запрос:

```
USE AdventureWorks2008R2;  
GO  
SELECT ProductID, Total = SUM(LineTotal)  
FROM Sales.SalesOrderDetail  
GROUP BY ProductID  
HAVING SUM(LineTotal) > $2000000.00;  
GO
```

Если необходимо убедиться в том, что в вычислениях для каждого продукта используется не менее 1500 элементов, используется инструкция HAVING COUNT(\*) > 1500, которая удаляет строки для продуктов с объемами продаж менее 1500. Запрос выглядит следующим образом:

```
USE AdventureWorks2008R2;  
GO  
SELECT ProductID, SUM(LineTotal) AS Total  
FROM Sales.SalesOrderDetail  
GROUP BY ProductID  
HAVING COUNT(*) > 1500;  
GO
```

## О. Вычисление общей суммы значений с помощью ключевого слова COMPUTE BY

В следующем примере приводятся два примера кода, поясняющих работу COMPUTE BY. В первом примере кода используется оператор COMPUTE BY с одной агрегатной функцией, а во втором — оператор COMPUTE BY с двумя агрегатными функциями.

В данном запросе вычисляется сумма заказов для каждого типа продукта, цены на который не превышают \$5.00.

```
USE AdventureWorks2008R2;  
GO  
SELECT ProductID, LineTotal  
FROM Sales.SalesOrderDetail  
WHERE UnitPrice < $5.00  
ORDER BY ProductID, LineTotal  
COMPUTE SUM(LineTotal) BY ProductID;  
GO
```

В данном запросе извлекаются типы и объемы продаж продуктов, цены на которые не превышают \$5.00. В предложении COMPUTE BY используются две различные агрегатные функции.

```
USE AdventureWorks2008R2;
GO
SELECT ProductID, LineTotal
FROM Sales.SalesOrderDetail
WHERE UnitPrice < $5.00
ORDER BY ProductID, LineTotal
COMPUTE SUM(LineTotal), MAX(LineTotal) BY ProductID;
GO
```

## **П. Вычисление итоговых значений с помощью ключевого слова COMPUTE без ключевого слова BY**

Ключевое слово COMPUTE без использования ключевого слова BY может быть использовано для вычисления общих сумм, общего количества и т.п.

В следующем примере вычисляется общая сумма цен, а также выводятся все авансовые платежи для всех типов продуктов с ценами ниже \$2.00.

```
USE AdventureWorks2008R2;
GO
SELECT ProductID, OrderQty, UnitPrice, LineTotal
FROM Sales.SalesOrderDetail
WHERE UnitPrice < $2.00
COMPUTE SUM(OrderQty), SUM(LineTotal);
GO
```

Оператор COMPUTE BY и оператор COMPUTE без ключевого слова BY могут использоваться одновременно в одном запросе. В следующем запросе вычисляется объем продаж и суммы для каждого продукта в отдельности, а затем находится общее количество продаж и вырученная сумма.

```
USE AdventureWorks2008R2;
GO
SELECT ProductID, OrderQty, UnitPrice, LineTotal
FROM Sales.SalesOrderDetail
WHERE UnitPrice < $5.00
ORDER BY ProductID
COMPUTE SUM(OrderQty), SUM(LineTotal) BY ProductID
COMPUTE SUM(OrderQty), SUM(LineTotal);
GO
```

## **Р. Вычисление сумм по всем строкам**

В следующем примере в списке выборки выводятся только три столбца, а затем вычисляются общие показатели, основанные на всех заказах во всех строках.

```
USE AdventureWorks2008R2;
GO
SELECT ProductID, OrderQty, LineTotal
FROM Sales.SalesOrderDetail
COMPUTE SUM(OrderQty), SUM(LineTotal);
GO
```

### С. Использование нескольких предложений COMPUTE

В следующем примере вычисляется сумма цен для всех заказов продуктов, цена которых не превышает \$5, а затем производится их сортировка по коду продукта и объему заказа. Суммы цен для всех заказов менее \$5 сортируются только по идентификатору. Можно использовать различные агрегатные функции в одной инструкции. Для этого необходимо использовать несколько предложений COMPUTE BY.

```
USE AdventureWorks2008R2;
GO
SELECT ProductID, OrderQty, UnitPrice, LineTotal
FROM Sales.SalesOrderDetail
WHERE UnitPrice < $5.00
ORDER BY ProductID, OrderQty, LineTotal
COMPUTE SUM(LineTotal) BY ProductID, OrderQty
COMPUTE SUM(LineTotal) BY ProductID;
GO
```

### Т. Сравнение GROUP BY и COMPUTE

В первом из представленных ниже примеров для вычисления суммы всех заказов на продукты, цена которых не превышает \$5.00, и группировки их по типу продукта используется предложение COMPUTE. Во втором примере выдача тех же сводных данных происходит с помощью оператора GROUP BY.

```
USE AdventureWorks2008R2;
GO
SELECT ProductID, LineTotal
FROM Sales.SalesOrderDetail
WHERE UnitPrice < $5.00
ORDER BY ProductID
COMPUTE SUM(LineTotal) BY ProductID;
GO
```

Это второй пример, в котором используется оператор GROUP BY.

```
USE AdventureWorks2008R2;
GO
SELECT ProductID, SUM(LineTotal) AS Total
FROM Sales.SalesOrderDetail
WHERE UnitPrice < $5.00
GROUP BY ProductID
ORDER BY ProductID;
GO
```

### У. Использование SELECT с предложениями GROUP BY, COMPUTE и ORDER BY

В следующем примере выводятся заказы на продукты с ценой менее \$5, а затем вычисляется сумма продаж для каждого продукта в отдельности и общая сумма по всем продуктам. Все вычисляемые столбцы отображаются в списке выборки.

```
USE AdventureWorks2008R2;
GO
SELECT ProductID, OrderQty, SUM(LineTotal) AS Total
FROM Sales.SalesOrderDetail
WHERE UnitPrice < $5.00
GROUP BY ProductID, OrderQty
ORDER BY ProductID, OrderQty
COMPUTE SUM(SUM(LineTotal)) BY ProductID, OrderQty
COMPUTE SUM(SUM(LineTotal));
```



GO

#### Ф. Использование подсказок оптимизатора INDEX

В следующем примере показаны два способа использования подсказок оптимизатора INDEX. В первом примере показано, как настроить оптимизатор на использование некластеризованного индекса для получения строк из таблицы. Во втором примере при использовании индекса 0 запускается просмотр таблицы.

```
USE AdventureWorks2008R2;
GO
SELECT pp.FirstName, pp.LastName, e.NationalIDNumber
FROM HumanResources.Employee AS e WITH (INDEX(AK_Employee_NationalIDNumber))
JOIN Person.Person AS pp ON e.BusinessEntityID = pp.BusinessEntityID
WHERE LastName = 'Johnson';
GO
```

```
-- Force a table scan by using INDEX = 0.
USE AdventureWorks2008R2;
GO
SELECT pp.LastName, pp.FirstName, e.JobTitle
FROM HumanResources.Employee AS e WITH (INDEX = 0) JOIN Person.Person AS pp
ON e.BusinessEntityID = pp.BusinessEntityID
WHERE LastName = 'Johnson';
GO
```

#### Х. Использование подсказок для операторов OPTION и GROUP

В следующем примере поясняется совместное использование предложений OPTION (GROUP) и GROUP BY.

```
USE AdventureWorks2008R2;
GO
SELECT ProductID, OrderQty, SUM(LineTotal) AS Total
FROM Sales.SalesOrderDetail
WHERE UnitPrice < $5.00
GROUP BY ProductID, OrderQty
ORDER BY ProductID, OrderQty
OPTION (HASH GROUP, FAST 10);
GO
```

#### Ц. Использование подсказок в запросе UNION

В следующем примере используется подсказка в запросе MERGE UNION.

```
USE AdventureWorks2008R2;
GO
SELECT *
FROM HumanResources.Employee AS e1
UNION
SELECT *
FROM HumanResources.Employee AS e2
OPTION (MERGE UNION);
GO
```

#### Ч. Использование одиночного предложения UNION

При выполнении следующего примера в результирующий набор включается содержимое столбцов ProductModelID и Name таблиц ProductModel и Gloves.

```
USE AdventureWorks2008R2;
GO
```

```
IF OBJECT_ID ('dbo.Gloves', 'U') IS NOT NULL
DROP TABLE dbo.Gloves;
GO
-- Create Gloves table.
SELECT ProductModelID, Name
INTO dbo.Gloves
FROM Production.ProductModel
WHERE ProductModelID IN (3, 4);
GO

-- Here is the simple union.
USE AdventureWorks2008R2;
GO
SELECT ProductModelID, Name
FROM Production.ProductModel
WHERE ProductModelID NOT IN (3, 4)
UNION
SELECT ProductModelID, Name
FROM dbo.Gloves
ORDER BY Name;
GO
```

### Ш. Использование SELECT INTO с предложением UNION

При выполнении следующего примера предложение INTO во второй инструкции SELECT указывает, что в таблице с именем ProductResults содержится итоговый результирующий набор объединения заданных столбцов таблиц ProductModel и Gloves. Заметим, что таблица Gloves была создана в результате выполнения первой инструкции SELECT.

```
USE AdventureWorks2008R2;
GO
IF OBJECT_ID ('dbo.ProductResults', 'U') IS NOT NULL
DROP TABLE dbo.ProductResults;
GO
IF OBJECT_ID ('dbo.Gloves', 'U') IS NOT NULL
DROP TABLE dbo.Gloves;
GO
-- Create Gloves table.
SELECT ProductModelID, Name
INTO dbo.Gloves
FROM Production.ProductModel
WHERE ProductModelID IN (3, 4);
GO

USE AdventureWorks2008R2;
GO
SELECT ProductModelID, Name
INTO dbo.ProductResults
FROM Production.ProductModel
WHERE ProductModelID NOT IN (3, 4)
UNION
SELECT ProductModelID, Name
FROM dbo.Gloves;
GO

SELECT *
FROM dbo.ProductResults;
```

**Щ. Использование предложения UNION для двух инструкций SELECT с предложением ORDER BY**

При использовании предложения UNION необходимо соблюдать порядок следования определенных параметров. В следующем примере приведены случаи правильного и неправильного использования предложения UNION с двумя инструкциями SELECT, при котором выходные столбцы должны быть переименованы.

```
USE AdventureWorks2008R2;
GO
IF OBJECT_ID ('dbo.Gloves', 'U') IS NOT NULL
DROP TABLE dbo.Gloves;
GO
-- Create Gloves table.
SELECT ProductModelID, Name
INTO dbo.Gloves
FROM Production.ProductModel
WHERE ProductModelID IN (3, 4);
GO
```

```
/* INCORRECT */
USE AdventureWorks2008R2;
GO
SELECT ProductModelID, Name
FROM Production.ProductModel
WHERE ProductModelID NOT IN (3, 4)
ORDER BY Name
UNION
SELECT ProductModelID, Name
FROM dbo.Gloves;
GO
```

```
/* CORRECT */
USE AdventureWorks2008R2;
GO
SELECT ProductModelID, Name
FROM Production.ProductModel
WHERE ProductModelID NOT IN (3, 4)
UNION
SELECT ProductModelID, Name
FROM dbo.Gloves
ORDER BY Name;
GO
```

**Ы. Использование предложения UNION с тремя инструкциями SELECT: пояснение влияния скобок и ключевого слова ALL**

В следующих примерах предложение UNION используется для комбинирования результатов из трех таблиц, содержащих по 5 одинаковых строк данных. В первом примере используется предложение UNION ALL, в результате чего выдаются все 15 строк. Во втором примере предложение UNION используется без ключевого слова ALL, что позволяет удалить повторяющиеся строки из комбинированного результата выполнения трех инструкций SELECT и вывести только 5 строк.

В третьем примере с первым предложением UNION используется ключевое слово ALL, а во втором предложении UNION вместо ключевого слова ALL используются скобки. Сначала выполняется второе предложение UNION, которое заключено в скобки. В результате возвращаются 5 строк, так как параметр ALL не используется и все

повторяющиеся строки удаляются. Полученные 5 строк совмещаются с результатами выполнения первой инструкции SELECT с помощью ключевого слова UNION ALL. В данном случае повторяющиеся строки двух множеств не удаляются. Окончательный результат состоит из 10 строк.

```
USE AdventureWorks2008R2;
GO
IF OBJECT_ID ('dbo.EmployeeOne', 'U') IS NOT NULL
DROP TABLE dbo.EmployeeOne;
GO
IF OBJECT_ID ('dbo.EmployeeTwo', 'U') IS NOT NULL
DROP TABLE dbo.EmployeeTwo;
GO
IF OBJECT_ID ('dbo.EmployeeThree', 'U') IS NOT NULL
DROP TABLE dbo.EmployeeThree;
GO

SELECT pp.LastName, pp.FirstName, e.JobTitle
INTO dbo.EmployeeOne
FROM Person.Person AS pp JOIN HumanResources.Employee AS e
ON e.BusinessEntityID = pp.BusinessEntityID
WHERE LastName = 'Johnson';
GO
SELECT pp.LastName, pp.FirstName, e.JobTitle
INTO dbo.EmployeeTwo
FROM Person.Person AS pp JOIN HumanResources.Employee AS e
ON e.BusinessEntityID = pp.BusinessEntityID
WHERE LastName = 'Johnson';
GO
SELECT pp.LastName, pp.FirstName, e.JobTitle
INTO dbo.EmployeeThree
FROM Person.Person AS pp JOIN HumanResources.Employee AS e
ON e.BusinessEntityID = pp.BusinessEntityID
WHERE LastName = 'Johnson';
GO
-- Union ALL
SELECT LastName, FirstName, JobTitle
FROM dbo.EmployeeOne
UNION ALL
SELECT LastName, FirstName, JobTitle
FROM dbo.EmployeeTwo
UNION ALL
SELECT LastName, FirstName, JobTitle
FROM dbo.EmployeeThree;
GO

SELECT LastName, FirstName, JobTitle
FROM dbo.EmployeeOne
UNION
SELECT LastName, FirstName, JobTitle
FROM dbo.EmployeeTwo
UNION
SELECT LastName, FirstName, JobTitle
FROM dbo.EmployeeThree;
GO

SELECT LastName, FirstName, JobTitle
FROM dbo.EmployeeOne
UNION ALL
(
```

```
SELECT LastName, FirstName, JobTitle
FROM dbo.EmployeeTwo
UNION
SELECT LastName, FirstName, JobTitle
FROM dbo.EmployeeThree
);
GO
```

## COMPUTE (Transact-SQL)

Формирует итоги, которые появляются в дополнительном столбце сводки в конце результирующего набора. При использовании с ключевым словом BY предложение COMPUTE формирует в результирующем наборе сегменты и промежуточные итоги. В одном запросе можно указать одновременно COMPUTE BY и COMPUTE.

### Важно!

В следующей версии Microsoft SQL Server эта возможность будет удалена. Не используйте ее при работе над новыми приложениями и как можно быстрее измените приложения, в которых она в настоящее время используется. Вместо этого используйте инструкцию ROLLUP. Дополнительные сведения см. в разделе [GROUP BY \(Transact-SQL\)](#).

### Синтаксис

```
[ COMPUTE
  { { AVG | COUNT | MAX | MIN | STDEV | STDEVP | VAR | VARP | SUM }
    ( expression ) } [ ,...n ]
  [ BY expression [ ,...n ] ]
]
```

### Аргументы

AVG | COUNT | MAX | MIN | STDEV | STDEVP | VAR | VARP | SUM

Указывает агрегат для выполнения. В предложении COMPUTE используются следующие агрегатные функции для строк.

| Агрегатная функция для строк | Результат                                                         |
|------------------------------|-------------------------------------------------------------------|
| AVG                          | Среднее значение в числовом выражении                             |
| COUNT                        | Количество выбранных строк                                        |
| MAX                          | Наибольшее значение в выражении                                   |
| MIN                          | Наименьшее значение в выражении                                   |
| STDEV                        | Статистическое стандартное отклонение для всех значений выражения |
| STDEVP                       | Статистическое стандартное отклонение для заполнения              |

по всем значениям выражения

|      |                                                                     |
|------|---------------------------------------------------------------------|
| SUM  | Сумма значений в числовом выражении                                 |
| VAR  | Статистическая дисперсия для всех значений выражения                |
| VARP | Статистическая дисперсия для заполнения по всем значениям выражения |

Эквивалента функции COUNT(\*) нет. Для нахождения сводных сведений, полученных при помощи предложений GROUP BY и COUNT(\*), используйте предложение COMPUTE без ключевого слова BY.

Эти функции не учитывают значения NULL.

Ключевое слово DISTINCT не применимо с агрегатными функциями для строк, определенными в предложении COMPUTE.

При сложении или нахождении среднего значения целочисленных данных компонент SQL Server Database Engine рассматривает результат как значение типа int, даже если в столбце содержатся данные типов smallint или tinyint. Дополнительные сведения о типах возвращаемых данных операций сложения или нахождения среднего значения см. в разделах [SUM \(Transact-SQL\)](#) и [AVG \(Transact-SQL\)](#).

#### Примечание

Для уменьшения вероятности возникновения ошибки переполнения в программах, использующих протокол ODBC или DB-Library, сделайте int типом всех переменных, объявленных для результатов сложения или нахождения среднего значения.

expression

Выражение [Выражения \(Transact-SQL\)](#), такое, как имя столбца, над которым выполняется вычисление. Аргумент expression должен присутствовать в списке выбора и быть указан идентично одному из выражений в списке выбора. Псевдоним столбца, определенный в списке выбора, не может быть использован в аргументе expression.

#### Примечание

Типы данных ntext, text или image не могут быть указаны в предложении COMPUTE или COMPUTE BY.

BY expression

Формирует сегменты и промежуточные итоги в результирующем наборе. Аргумент expression идентичен выражению order\_by\_expression в связанном предложении ORDER BY. Обычно это имя или псевдоним столбца. Можно указать несколько выражений. При перечислении нескольких выражений после ключевого слова BY группа разбивается на подгруппы, а агрегатные функции применяются к каждому

уровню группирования.

При использовании предложения COMPUTE BY необходимо также использовать предложение [ORDER BY](#). Выражения должны быть такими же, как в предложении ORDER BY, или быть их подмножеством. Располагаться они должны в той же последовательности. Например, если предложением ORDER BY является ORDER BY a, b, c, предложение COMPUTE может быть любым из следующих:

```
COMPUTE BY a, b, c
COMPUTE BY a, b
COMPUTE BY a
```

### Примечание

В инструкции SELECT с предложением COMPUTE порядок столбцов в списке выбора переопределяет порядок агрегатных функций в предложении COMPUTE. Программисты, использующие протоколы ODBC и DB-Library, должны знать об этом требовании к порядку расположения результатов агрегатных функций в правильном месте.

Использовать предложение COMPUTE в инструкции SELECT INTO нельзя, потому что инструкции, включающие предложение COMPUTE, формируют таблицы и их итоговые результаты не сохраняются в базе данных. Таким образом, любые вычисления, производимые в предложении COMPUTE, не появляются в новой таблице, созданной с помощью инструкции SELECT INTO.

Предложение COMPUTE нельзя использовать, если инструкция SELECT является частью инструкции DECLARE CURSOR.

### Примеры

#### ***А. Использование предложения COMPUTE в запросе для возвращения итогов***

В следующем примере инструкция SELECT использует простое предложение COMPUTE для формирования основной итоговой суммы по столбцам SubTotal и TotalDue из таблицы SalesOrderHeader.

```
USE AdventureWorks2008R2;
GO
SELECT CustomerID, OrderDate, SubTotal, TotalDue
FROM Sales.SalesOrderHeader
WHERE SalesPersonID = 35
ORDER BY OrderDate
COMPUTE SUM(SubTotal), SUM(TotalDue);
```

#### ***Б. Использование предложения COMPUTE в запросе для возвращения итогов***

В следующем примере инструкция SELECT использует предложение COMPUTE для формирования по каждому менеджеру по продажам итоговой суммы по столбцам SubTotal и TotalDue из таблицы SalesOrderHeader.

```
USE AdventureWorks2008R2;
GO
SELECT SalesPersonID, CustomerID, OrderDate, SubTotal, TotalDue
FROM Sales.SalesOrderHeader
```

```
ORDER BY SalesPersonID, OrderDate
COMPUTE SUM(SubTotal), SUM(TotalDue) BY SalesPersonID;
```

## Предложение FOR (Transact-SQL)

Предложение FOR используется для задания параметра BROWSE или XML. Параметры BROWSE и XML не связаны друг с другом.

### Важно!

Директива XMLDATA для параметра XML FOR является устаревшей. В режимах RAW и AUTO следует использовать создание XSD-схем. В режиме EXPLICIT для директивы XMLDATA замены нет. В будущей версии Microsoft SQL Server эта возможность будет удалена. Избегайте использования этой возможности в новых разработках и запланируйте изменение существующих приложений, в которых она применяется.

### Синтаксис

```
[ FOR { BROWSE | <XML> } ]
<XML> ::=
XML
{
    { RAW [ ( 'ElementName' ) ] | AUTO }
    [
        <CommonDirectives>
        [ , { XMLDATA | XMLSCHEMA [ ( 'TargetNameSpaceURI' ) ] } ]
        [ , ELEMENTS [ XSINIL | ABSENT ] ]
    ]
| EXPLICIT
    [
        <CommonDirectives>
        [ , XMLDATA ]
    ]
| PATH [ ( 'ElementName' ) ]
    [
        <CommonDirectives>
        [ , ELEMENTS [ XSINIL | ABSENT ] ]
    ]
}

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

### Аргументы

#### BROWSE

Активирует возможность обновления данных во время их просмотра с помощью курсора в режиме обзора DB-Library. Таблицу можно просмотреть внутри приложения, если в таблице содержится столбец timestamp, если таблице присвоен уникальный индекс или если в конце инструкции SELECT, отсылаемой экземпляру SQL Server, имеется параметр FOR BROWSE.

### Примечание



Нельзя использовать синтаксис <lock\_hint> HOLDLOCK для инструкции SELECT, включающей в себя параметр FOR BROWSE.

Параметр FOR BROWSE не может быть использован в инструкциях SELECT, соединенных оператором UNION.

### Примечание

Если ключевые столбцы уникального индекса таблицы могут принимать неопределенные значения, а таблица находится внутри внешнего соединения, индексы в режиме обзора не поддерживаются.

Режим просмотра позволяет просматривать строки в таблице SQL Server и обновлять данные в таблице по одной строке одновременно. Чтобы получить доступ к таблице SQL Server в приложении в режиме просмотра, необходимо использовать одну из следующих двух функций.

- Инструкция SELECT, применяемая для получения доступа к данным таблицы SQL Server, должна оканчиваться ключевыми словами FOR BROWSE. Если для использования режима просмотра включен параметр FOR BROWSE, создаются временные таблицы.
- Необходимо выполнить следующую инструкцию Transact-SQL, чтобы включить режим просмотра с параметром NO\_BROWSETABLE:

```
SET NO_BROWSETABLE ON
```

После включения параметра NO\_BROWSETABLE все инструкции SELECT действуют так, как если бы к инструкциям был добавлен параметр FOR BROWSE. Однако параметр NO\_BROWSETABLE не создает временные таблицы, которые обычно используются параметром FOR BROWSE, чтобы передать результаты в приложение.

Если предпринимается попытка получить доступ к данным таблиц SQL Server в режиме просмотра с помощью запроса SELECT, содержащего инструкцию внешнего соединения и если определен уникальный индекс на таблице, которая присутствует во внутренней части инструкции внешнего соединения, в режиме просмотра не поддерживается уникальный индекс. В режиме просмотра уникальный индекс поддерживается, только если все ключевые столбцы уникального индекса могут принимать значения NULL. Уникальный индекс не поддерживается в режиме просмотра, если следующие условия являются истинными.

- Предпринимается попытка получить доступ к данным таблиц SQL Server в режиме просмотра с использованием запроса SELECT, содержащего инструкцию внешнего соединения.
- Уникальный индекс определен на таблице, которая присутствует во внутренней части инструкции внешнего соединения.

Чтобы воспроизвести это поведение в режиме просмотра, выполните следующие шаги.

1. В среде Среда SQL Server Management Studio создайте базу данных с именем SampleDB.
2. В базе данных SampleDB создайте таблицы tleft и tright так, чтобы каждая содержала один столбец с именем c1. Определите уникальный индекс на столбце c1 в таблице tleft и предусмотрите, чтобы этот столбец принимал значения NULL. Чтобы это сделать, выполните в соответствующем окне запроса следующие инструкции Transact-SQL:

```
CREATE TABLE tleft(c1 INT NULL UNIQUE) ;  
GO  
CREATE TABLE tright(c1 INT NULL) ;  
GO
```

3. Вставьте несколько значений в таблицу tleft и таблицу tright. Обязательно вставьте значение NULL в таблицу tleft. Чтобы это сделать, выполните в окне запроса следующие инструкции Transact-SQL:

```
INSERT INTO tleft VALUES(2) ;  
INSERT INTO tleft VALUES(NULL) ;  
INSERT INTO tright VALUES(1) ;  
INSERT INTO tright VALUES(3) ;  
INSERT INTO tright VALUES(NULL) ;  
GO
```

4. Включите параметр NO\_BROWSETABLE. Чтобы это сделать, выполните в окне запроса следующие инструкции Transact-SQL:

```
SET NO_BROWSETABLE ON ;  
GO
```

5. Получите доступ к данным в таблице tleft и таблице tright с помощью инструкции внешнего соединения в запросе SELECT. Убедитесь, что таблица tleft находится во внутренней части инструкции внешнего соединения. Чтобы это сделать, выполните в окне запроса следующие инструкции Transact-SQL:

```
SELECT tleft.c1  
FROM tleft  
RIGHT JOIN tright  
ON tleft.c1 = tright.c1  
WHERE tright.c1 <> 2 ;
```

Обратите внимание на следующие выходные данные в области «Результаты»:

c1

----

NULL

NULL

После выполнения запроса SELECT для получения доступа к таблицам в режиме

просмотра результирующий набор запроса SELECT содержит два значения NULL для столбца c1 в таблице tleft, поскольку таково определение инструкции правого внешнего соединения. Поэтому в результирующем наборе невозможно различить значения NULL, полученные из таблицы, и значения NULL, добавленные инструкцией правого внешнего соединения. Могут быть получены неверные результаты, если необходимо пропустить значения NULL из результирующего набора.

### Примечание

Если столбцы, которые включены в уникальный индекс, не допускают значения NULL, это значит, что все значения NULL в результирующем наборе были добавлены инструкцией правого внешнего соединения.

## XML

Задаёт возврат результатов запроса в виде XML-документа. Должен быть задан один из следующих режимов XML: RAW, AUTO, EXPLICIT. Дополнительные сведения об XML-данных и Службы Analysis Services см. в разделе [Создание XML с помощью предложения FOR XML](#).

### RAW [ ('ElementName') ]

Получает результат запроса и преобразует каждую строку результирующего набора в элемент XML, для которого в качестве тега используется общий идентификатор <row />. Дополнительно можно задать имя для элемента строки. Для результирующего выхода в формате XML в качестве элементов, создаваемых для каждой строки, используются определенные ElementName . Дополнительные сведения см. в разделах [Использование режима RAW](#) и [Использование режима RAW](#).

## AUTO

Возвращает результаты запроса в виде простого вложенного дерева XML. Каждая таблица предложения FROM, для которой в предложении SELECT приведен хотя бы один столбец, отображается как элемент XML. Столбцы, перечисленные в предложении SELECT, сопоставлены с соответствующими атрибутами элемента. Дополнительные сведения см. в разделе [Использование режима AUTO](#).

## EXPLICIT

Задаёт явное определение формы результирующего XML-дерева. При использовании данного режима запросы должны записываться таким образом, чтобы дополнительные сведения о вложениях могли быть заданы явно. Дополнительные сведения см. в разделе [Использование режима EXPLICIT](#).

## XMLDATA

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

### XMLSCHEMA [ ('TargetNamespaceURI') ]

Возвращает встроенную XSD-схему. При задании указанной директивы, возвращающей заданное пространство имен схемы, дополнительно можно задать URI целевого пространства имен. Дополнительные сведения см. в разделе [Создание встроенных XSD-схем](#).

## ELEMENTS

Задаёт возврат столбцов в виде вложенных элементов. В противном случае столбцы будут сопоставлены с XML-атрибутами. Данный параметр поддерживается только в режимах RAW, AUTO и PATH. Дополнительные сведения см. в разделе [Использование режима RAW](#).

## XSINIL

Задаёт создание элемента с атрибутом **xsi:nil**, установленного в значение **True**, для столбцов со значениями NULL. Данный параметр может быть указан только в директиве ELEMENTS. Дополнительные сведения см. в разделе [Создание элементов для значений NULL с помощью параметра XSINIL](#).

## ABSENT

Указывает, что соответствующие XML-элементы для столбцов со значениями NULL к XML-результату не добавляются. Указывайте данный параметр только с директивой ELEMENTS.

## PATH [ ('ElementName') ]

Создаёт упаковщик элементов <строки> для каждой строки в результирующем наборе. Для упаковщика элементов <строки> можно дополнительно задать имя элемента. При задании пустой строки, например FOR XML PATH ('') ), упаковщик элементов не создается. Использование директивы PATH даёт более простой способ написания запросов, чем написание запросов с помощью директивы EXPLICIT. Дополнительные сведения см. в разделе [Использование режима PATH](#).

## BINARY BASE64

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

## TYPE

Задаёт следующий формат выдаваемых запросом данных: тип **xml**. Дополнительные сведения см. в разделе [Директива TYPE в запросах FOR XML](#).

## ROOT [ ('RootName') ]

Задаёт добавление единичного элемента высшего уровня к результирующему XML-документу. Дополнительно можно указать имя корневого элемента, который необходимо сформировать. Если имя корневого элемента не задано, то добавляется <корневой> элемент по умолчанию.

## Примеры

В данном примере задается параметр FOR XML AUTO с параметрами TYPE и XMLSCHEMA. Благодаря параметру TYPE результирующий набор возвращается клиенту в формате **xml**. Параметр XMLSCHEMA определяет встроенную XSD-схему, включаемую в возвращаемые XML-данные, а параметр ELEMENTS указывает, что результаты в формате XML основываются на элементах.

```
USE AdventureWorks2008R2;
GO
SELECT p.BusinessEntityID, FirstName, LastName, PhoneNumber AS Phone
FROM Person.Person AS p
Join Person.PersonPhone AS pph ON p.BusinessEntityID = pph.BusinessEntityID
WHERE LastName LIKE 'G%'
ORDER BY LastName, FirstName
FOR XML AUTO, TYPE, XMLSCHEMA, ELEMENTS XSINIL;
```

## GROUP BY (Transact-SQL)

Группирует выбранный набор строк для получения набора сводных строк по значениям одного или нескольких столбцов или выражений в SQL Server 2008 R2. Возвращается одна строка для каждой группы. Агрегатные функции в списке <select> предложения SELECT предоставляют сведения о каждой группе, а не об отдельных строках.

Предложение GROUP BY имеет два синтаксиса: совместимый с ISO и несовместимый с ISO. В каждой отдельной инструкции SELECT может использоваться только один стиль синтаксиса. Во всех новых разработках используйте совместимый с ISO синтаксис. Синтаксис, несовместимый с ISO, служит для обеспечения обратной совместимости.

В этом разделе предложение GROUP BY можно описать как общее или простое.

- Общее предложение GROUP BY включает конструкции GROUPING SETS, CUBE, ROLLUP, WITH CUBE и WITH ROLLUP.
- Простое предложение GROUP BY не включает конструкции GROUPING SETS, CUBE, ROLLUP, WITH CUBE и WITH ROLLUP. Предложение GROUP BY (), предназначенное для определения общего итога, рассматривается как простое предложение GROUP BY.

## Синтаксис

ISO-Compliant Syntax

GROUP BY <group by spec>

<group by spec> ::=  
    <group by item> [ , ...n ]

<group by item> ::=  
    <simple group by item>  
    | <rollup spec>  
    | <cube spec>  
    | <grouping sets spec>  
    | <grand total>

<simple group by item> ::=

```

<column_expression>

<rollup spec> ::=
    ROLLUP ( <composite element list> )<cube spec> ::=
    CUBE ( <composite element list> )<composite element list> ::=
    <composite element> [ ,...n ]

<composite element> ::=
    <simple group by item>
    | ( <simple group by item list> )<simple group by item list> ::=
    <simple group by item> [ ,...n ]

<grouping sets spec> ::=
    GROUPING SETS ( <grouping set list> )<grouping set list> ::=
    <grouping set> [ ,...n ]

<grouping set> ::=
    <grand total>
    | <grouping set item>
    | ( <grouping set item list> )<empty group> ::=()<grouping set item> ::=
    <simple group by item>
    | <rollup spec>
    | <cube spec>

<grouping set item list> ::=
    <grouping set item> [ ,...n ]

Non-ISO-Compliant Syntax
[ GROUP BY [ ALL ] group_by_expression [ ,...n ]
  [ WITH { CUBE | ROLLUP } ]
]

```

## Аргументы

<column\_expression>  
[Выражение](#), на основании которого выполняется операция группирования.

## ROLLUP ( )

Формирует статистические строки простого предложения GROUP BY и строки подытогов или строки со статистическими вычислениями высокого уровня, а также строки общего итога.

Количество возвращаемых группирований равно количеству выражений в <составном списке элементов> плюс один. Например, рассмотрим следующую инструкцию.

```

SELECT a, b, c, SUM ( <expression> )
FROM T
GROUP BY ROLLUP (a,b,c);

```

Для каждого уникального сочетания значений (a, b, c), (a, b) и (a) формируется одна строка с подытогом. Вычисляется также строка общего итога.

Столбцы свертываются справа налево. Последовательность расположения столбцов влияет на выходное группирование ROLLUP и может отразиться на количестве строк в результирующем наборе.

## CUBE ( )

Формирует статистические строки простого предложения GROUP BY, строки со статистическими вычислениями высокого уровня конструкции ROLLUP и строки с результатами перекрестных вычислений.

Выходные данные CUBE являются группированием для всех перестановок выражений в <составном списке элементов>.

Количество формируемых группирований равно  $(2^n)$ , где  $n$  — количество выражений в <составном списке элементов>. Например, рассмотрим следующую инструкцию.

```
SELECT a, b, c, SUM (<expression>)
FROM T
GROUP BY CUBE (a,b,c);
```

Формируется одна строка для каждого уникального сочетания значений (a, b, c), (a, b), (a, c), (b, c), (a), (b) и (c) с подытогом для каждой строки и строкой общего итога.

Выходные данные CUBE не зависят от порядка столбцов.

## GROUPING SETS ( )

Указывает несколько группирований данных в одном запросе. Выполняется статистическая обработка только указанных групп, а не полного набора статистических данных, формируемых с помощью конструкций CUBE или ROLLUP. Результаты эквивалентны тем, что формируются с применением конструкции UNION ALL к указанным группам. Конструкция GROUPING SETS может содержать единственный элемент или список элементов. В конструкции GROUPING SETS могут быть указаны группирования, эквивалентные тем, которые возвращает конструкция ROLLUP или CUBE. Примеры см. в разделе [Эквиваленты GROUPING SETS](#). <Список элементов группирующего набора> может содержать конструкцию ROLLUP или CUBE.

## ( )

Пустая группа формирует итог.

## Синтаксис, несовместимый с ISO

### ALL

В будущей версии Microsoft SQL Server эта возможность будет удалена. Избегайте использования этой возможности в новых разработках и запланируйте изменение существующих приложений, в которых она применяется. Включает все группы и результирующие наборы (даже не имеющие строк), которые удовлетворяют условию поиска, указанному в предложении WHERE. Если задан аргумент ALL, для сводных столбцов групп, не удовлетворяющие условию поиска, возвращаются значения NULL. Аргумент ALL нельзя указывать с операторами CUBE или ROLLUP.

Предложение GROUP BY ALL не поддерживается в запросах с доступом к

удаленным таблицам, если в запросе присутствует также предложение WHERE. Применение предложения GROUP BY ALL к столбцам, имеющим атрибут FILESTREAM, приведет к ошибке.

group\_by\_expression

Выражение, по которому выполняется группирование. Аргумент group\_by\_expression также называется столбцом группирования. Аргумент group\_by expression может быть столбцом или нестатистическим выражением, которое ссылается на столбец, возвращаемый предложением FROM. Псевдоним столбца, определенный в списке SELECT, нельзя использовать при указании столбца группирования.

### Примечание

Столбцы типа text, ntext и image нельзя использовать в аргументе group\_by\_expression.

Для предложений GROUP BY, не содержащих операторов CUBE или ROLLUP, количество элементов group\_by\_expression ограничивается размером столбцов GROUP BY, статистически обрабатываемых столбцов и статистических значений, включенных в запрос. Это объясняется ограничением размера промежуточной рабочей таблицы (8 060 байт), необходимой для хранения промежуточных результатов запроса. При указании CUBE или ROLLUP максимально разрешенное количество выражений группирования равно 12.

Методы для типа данных xml нельзя указывать непосредственно в аргументе group\_by\_expression. Вместо этого следует создать ссылку на пользовательскую функцию, которая включает методы для типа данных xml или ссылается на вычисляемый столбец, использующий эти методы.

### WITH CUBE

В будущей версии Microsoft SQL Server эта возможность будет удалена. Избегайте использования этой возможности в новых разработках и запланируйте изменение существующих приложений, в которых она применяется. Указывает, что помимо строк, предоставляемых GROUP BY, в результирующий набор включаются сводные строки. Сводная строка GROUP BY возвращается для всех возможных сочетаний групп и подгрупп в результирующем наборе. Чтобы определить, представляют ли значения NULL в результирующем наборе сводные значения GROUP BY, используйте функцию GROUPING.

Количество сводных строк в результирующем наборе определяется по количеству столбцов, включенных в предложение GROUP BY. Поскольку оператор CUBE возвращает все возможные сочетания групп и подгрупп, количество строк остается тем же, независимо от заданного порядка группирования столбцов.

### WITH ROLLUP

В будущей версии Microsoft SQL Server эта возможность будет удалена. Избегайте использования этой возможности в новых разработках и запланируйте изменение существующих приложений, в которых она применяется. Указывает, что помимо строк, предоставляемых GROUP BY, в результирующий набор включаются сводные



строки. Группы обобщаются в иерархическом порядке, начиная с самого нижнего уровня в группе и заканчивая самым верхним. Иерархия группы определяется порядком, в котором заданы столбцы, по которым производится группирование. Изменение порядка столбцов, по которым производится группирование, может повлиять на количество строк в результирующем наборе.

### Важно!

Статистические вычисления с ключевым словом DISTINCT, например AVG(DISTINCT column\_name), COUNT(DISTINCT column\_name) и SUM(DISTINCT column\_name), при использовании CUBE и ROLLUP не поддерживаются. Если они используются, то компонент SQL Server Database Engine возвращает сообщение об ошибке и отменяет запрос.

### Замечания

Выражения в предложении GROUP BY могут содержать столбцы таблиц, производных таблиц или представлений из предложения FROM. Эти столбцы не обязательно должны появляться в списке <select> предложения SELECT.

Каждый столбец таблицы или представления в любом нестатистическом выражении в списке <select> должен быть включен в список GROUP BY.

- Следующие инструкции являются допустимыми.

```
SELECT ColumnA, ColumnB FROM T GROUP BY ColumnA, ColumnB;  
SELECT ColumnA + ColumnB FROM T GROUP BY ColumnA, ColumnB;  
SELECT ColumnA + ColumnB FROM T GROUP BY ColumnA + ColumnB;  
SELECT ColumnA + ColumnB + constant FROM T GROUP BY ColumnA, ColumnB;
```

- Следующие инструкции не являются допустимыми.

```
SELECT ColumnA, ColumnB FROM T GROUP BY ColumnA + ColumnB  
SELECT ColumnA + constant + ColumnB FROM T GROUP BY ColumnA + ColumnB;
```

Если в предложение SELECT <список выбора> включены агрегатные функции, инструкция GROUP BY вычисляет сводные значения для каждой группы. (Они известны как векторные статистические выражения.)

Строки, которые не соответствуют условиям в предложении WHERE, удаляются до выполнения любых операций группирования.

Предложение [HAVING](#) используется с предложением GROUP BY для фильтрации групп в результирующем наборе.

Применение предложения GROUP BY не упорядочивает результирующий набор. Чтобы упорядочить результирующий набор, необходимо использовать предложение ORDER BY.

Если столбец группирования содержит значения NULL, они рассматриваются как равные и помещаются в одну группу.

Нельзя использовать предложение GROUP BY с псевдонимом для замены имени столбца в предложении AS, если этот псевдоним не заменяет имя столбца в производной таблице в предложении FROM.

Повторяющиеся наборы группирования в списке GROUPING SETS не устраняются. Повторяющиеся наборы группирования могут быть сформированы, если выражение столбца указано больше одного раза или если входящее в список выражение столбца сформировано также конструкцией CUBE или ROLLUP в списке GROUPING SETS.

С конструкциями ROLLUP, CUBE и GROUPING SETS поддерживаются статистические функции с ключевым словом DISTINCT, например AVG (DISTINCT column\_name), COUNT (DISTINCT column\_name) и SUM (DISTINCT column\_name).

Конструкции ROLLUP, CUBE и GROUPING SETS не могут быть указаны в индексированном представлении.

Конструкции GROUP BY или HAVING не могут использоваться непосредственно для столбцов типа ntext, text или image. Эти столбцы могут использоваться в качестве аргументов в функциях, которые возвращают значение другого типа данных, таких как SUBSTRING() и CAST().

Методы данных типа xml не могут быть указаны непосредственно в аргументе <column\_expression>. Вместо этого следует создать ссылку на пользовательскую функцию, которая включает методы для типа данных xml или ссылается на вычисляемый столбец, использующий эти методы.

## **Ограничения предложения GROUP BY для конструкций GROUPING SETS, ROLLUP и CUBE**

### ***Ограничения синтаксиса***

Нельзя использовать конструкции GROUPING SETS в предложении GROUP BY, если они не входят в состав списка GROUPING SETS. Например, форма GROUP BY C1, (C2,..., Cn) недопустима, но форма GROUP BY GROUPING SETS (C1, (C2, ..., Cn)) является допустимой.

Нельзя использовать ключевые слова GROUPING SETS внутри самих конструкций GROUPING SETS. Например, выражение GROUP BY GROUPING SETS (C1, GROUPING SETS (C2, C3)) недопустимо.

В предложении GROUP BY с ключевыми словами ROLLUP, CUBE или GROUPING SETS нельзя использовать ключевые слова ALL, WITH CUBE и WITH ROLLUP, несовместимые с ISO.

### ***Ограничения размера***

Для простых предложений GROUP BY количество выражений не ограничено.

Для предложения GROUP BY, в котором используются ключевые слова ROLLUP, CUBE или GROUPING SETS, максимальное количество выражений равно 32, а максимальное количество группирующих наборов, которые могут быть созданы, равно 4096 (212). Следующие примеры завершаются неудачно, поскольку предложение GROUP BY является слишком сложным.

- В следующих примерах формируются 8192 (213) группирующих наборов.

```
GROUP BY CUBE (a1, ..., a13)
GROUP BY a1, ..., a13 WITH CUBE
```

- В следующем примере формируются 4097 (212 + 1) группирующих наборов.

```
GROUP BY GROUPING SETS( CUBE(a1, ..., a12), b )
```

- В следующем примере также формируются 4097 (212 + 1) группирующих наборов. И функция CUBE (), и группирующий набор () формируют строку общего итога, а повторяющиеся группирующие наборы не удаляются.

```
GROUP BY GROUPING SETS( CUBE(a1, ..., a12), ( ) )
```

### Поддержка функций предложения GROUP BY, совместимых с ISO и ANSI SQL-2006

В SQL Server 2008 и более поздних версиях предложение GROUP BY не может содержать вложенный запрос в выражении, используемом для группирования по списку. Возвращается ошибка 144.

SQL Server 2008 и более поздние версии поддерживают все функции предложения GROUP BY, которые включены в стандарт SQL-2006, со следующими синтаксическими исключениями.

- Применение группирующих наборов в предложении GROUP BY недопустимо, если они не составляют часть явного списка GROUPING SETS. Например, предложение GROUP BY Column1, (Column2, ...ColumnN) допускается в этом стандарте, но не в SQL Server. Можно использовать выражение GROUP BY C1, GROUPING SETS ((Column2, ...ColumnN)) или GROUP BY Column1, Column2, ... ColumnN. Эти инструкции семантически эквивалентны предыдущему примеру предложения GROUP BY. Это позволяет избежать возможности неправильной интерпретации предложения GROUP BY Column1, (Column2, ...ColumnN) как GROUP BY C1, GROUPING SETS ((Column2, ...ColumnN)). Эта инструкция не является семантически эквивалентной.
- Применение группирующих наборов внутри группирующих наборов не допускается. Например, предложение GROUP BY GROUPING SETS (A1, A2,...An, GROUPING SETS (C1, C2, ...Cn)) допустимо в стандарте SQL-2006, но не в SQL Server. В SQL Server 2008 и более поздних версиях можно использовать выражение GROUP BY GROUPING SETS( A1, A2,...An, C1, C2, ...Cn ) или GROUP BY GROUPING SETS( (A1), (A2), ... (An), (C1), (C2), ... (Cn) ). Эти примеры семантически эквивалентны первому примеру предложения GROUP BY и имеют более понятный синтаксис.
- Нельзя применять ключевые слова GROUP BY [ALL/DISTINCT] в общем предложении GROUP BY и в конструкциях GROUPING SETS, ROLLUP, CUBE, WITH CUBE или WITH ROLLUP. Ключевое слово ALL применяется по умолчанию и задано неявно.

### Сравнение поддерживаемых функций предложения GROUP BY

В следующей таблице описаны поддерживаемые функции предложения GROUP BY с учетом версии SQL Server и уровня совместимости базы данных.

| Функция                                                                                              | Службы SQL Server 2005<br>Integration Services                                                                                                                                               | Уровень совместимости<br>SQL Server 2008 — 100                                                                                                                                                             | Уровень совместимости<br>SQL Server 2008 — 90 или<br>меньше                                                                                                                               |
|------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Статистические<br>функции<br>DISTINCT                                                                | Не поддерживаются<br>для конструкций WITH<br>CUBE или WITH<br>ROLLUP.                                                                                                                        | Поддерживаются для<br>конструкций WITH<br>CUBE, WITH ROLLUP,<br>GROUPING SETS,<br>CUBE или ROLLUP.                                                                                                         | То же, что и уровень<br>совместимости SQL<br>Server 2008, равный<br>100.                                                                                                                  |
|                                                                                                      |                                                                                                                                                                                              | Определяемая<br>пользователем<br>функция <b>dbo.cube</b><br>(arg1,...argN) или<br><b>dbo.rollup</b> (arg1,...argN<br>) в предложении<br>GROUP BY<br>недопустима.                                           |                                                                                                                                                                                           |
|                                                                                                      |                                                                                                                                                                                              | Например:                                                                                                                                                                                                  |                                                                                                                                                                                           |
| Определяемая<br>пользователем<br>функция с<br>именем CUBE<br>или ROLLUP в<br>предложении<br>GROUP BY | Определяемая<br>пользователем<br>функция<br><b>dbo.cube</b> (arg1,...argN)<br>или<br><b>dbo.rollup</b> (arg1,...argN<br>) в предложении<br>GROUP BY является<br>допустимой.<br><br>Например: | <pre>SELECT SUM (x) FROM T GROUP BY dbo.cube(y);</pre> <p>Выдается следующее<br/>сообщение об ошибке:<br/>«Применение<br/>неверного синтаксиса<br/>недалеко от ключевого<br/>слова 'cube'   'rollup'».</p> | Определяемая<br>пользователем<br>функция <b>dbo.cube</b><br>(arg1,...argN) или<br><b>dbo.rollup</b> (arg1,...argN<br>) в предложении<br>GROUP BY является<br>допустимой.<br><br>Например: |
|                                                                                                      | <pre>SELECT SUM (x) FROM T GROUP BY dbo.cube(y);</pre>                                                                                                                                       | Чтобы избежать этой<br>проблемы, замените<br>конструкцию <b>dbo.cube</b><br>на [dbo].[cube] или<br>конструкцию<br><b>dbo.rollup</b> на [dbo].<br>[rollup].                                                 | <pre>SELECT SUM (x) FROM T GROUP BY dbo.cube(y);</pre>                                                                                                                                    |
|                                                                                                      |                                                                                                                                                                                              | Следующий пример<br>является допустимым:                                                                                                                                                                   |                                                                                                                                                                                           |
|                                                                                                      |                                                                                                                                                                                              | <pre>SELECT SUM (x) FROM T GROUP BY [dbo].[cube] (y);</pre>                                                                                                                                                |                                                                                                                                                                                           |
| GROUPING SETS                                                                                        | Не поддерживается                                                                                                                                                                            | Поддерживается                                                                                                                                                                                             | Поддерживается                                                                                                                                                                            |
| CUBE                                                                                                 | Не поддерживается                                                                                                                                                                            | Поддерживается                                                                                                                                                                                             | Не поддерживается                                                                                                                                                                         |

|                                                                                                            |                   |                |                   |
|------------------------------------------------------------------------------------------------------------|-------------------|----------------|-------------------|
| ROLLUP                                                                                                     | Не поддерживается | Поддерживается | Не поддерживается |
| Общий итог,<br>такой как<br>GROUP BY ()                                                                    | Не поддерживается | Поддерживается | Поддерживается    |
| Функция<br>GROUPING_ID                                                                                     | Не поддерживается | Поддерживается | Поддерживается    |
| Функция<br>GROUPING                                                                                        | Поддерживается    | Поддерживается | Поддерживается    |
| WITH CUBE                                                                                                  | Поддерживается    | Поддерживается | Поддерживается    |
| WITH ROLLUP                                                                                                | Поддерживается    | Поддерживается | Поддерживается    |
| Удаление<br>«повторяющегося<br>» группирования<br>с помощью<br>конструкции<br>WITH CUBE или<br>WITH ROLLUP | Поддерживается    | Поддерживается | Поддерживается    |

## Примеры

Примеры использования ключевых слов GROUPING SETS, ROLLUP и CUBE см. в разделе [Использование предложения GROUP BY с операторами ROLLUP, CUBE и GROUPING SETS](#).

### А. Использование простого предложения GROUP BY

В следующем примере происходит извлечение суммы для каждого SalesOrderID из таблицы SalesOrderDetail.

```
USE AdventureWorks2008R2;
GO
SELECT SalesOrderID, SUM(LineTotal) AS SubTotal
FROM Sales.SalesOrderDetail AS sod
GROUP BY SalesOrderID
ORDER BY SalesOrderID;
```

### Б. Использование предложения GROUP BY с несколькими таблицами

В следующем примере из таблицы Address, соединенной с таблицей EmployeeAddress, получается количество работников для каждого города City.

```
USE AdventureWorks2008R2;
GO
SELECT a.City, COUNT(bea.AddressID) AS EmployeeCount
FROM Person.BusinessEntityAddress AS bea
```

```
INNER JOIN Person.Address AS a
  ON bea.AddressID = a.AddressID
GROUP BY a.City
ORDER BY a.City;
```

### ***В. Использование предложения GROUP BY в выражениях***

В следующем примере показано, как извлечь данные об общем объеме продаж за каждый год с помощью функции DATEPART. Одно и то же выражение должно присутствовать как в списке SELECT, так и в предложении GROUP BY.

```
USE AdventureWorks2008R2;
GO
SELECT DATEPART(yyyy,OrderDate) AS N'Year'
      ,SUM(TotalDue) AS N'Total Order Amount'
FROM Sales.SalesOrderHeader
GROUP BY DATEPART(yyyy,OrderDate)
ORDER BY DATEPART(yyyy,OrderDate);
```

### ***Г. Использование предложения GROUP BY с предложением HAVING***

В следующем примере используется предложение HAVING, чтобы указать, какая из групп, сформированная в предложении GROUP BY, должна быть включена в результирующий набор.

```
USE AdventureWorks2008R2;
GO
SELECT DATEPART(yyyy,OrderDate) AS N'Year'
      ,SUM(TotalDue) AS N'Total Order Amount'
FROM Sales.SalesOrderHeader
GROUP BY DATEPART(yyyy,OrderDate)
HAVING DATEPART(yyyy,OrderDate) >= N'2003'
ORDER BY DATEPART(yyyy,OrderDate);
```

## **HAVING (Transact-SQL)**

Определяет условие поиска для группы или статистического выражения. Предложение HAVING можно использовать только в инструкции SELECT. Предложение HAVING обычно используется в предложении GROUP BY. Когда GROUP BY не используется, предложение HAVING работает так же, как и предложение WHERE.

### **Синтаксис**

```
[ HAVING <search condition> ]
```

### **Аргументы**

<search\_condition>

Определяет условие поиска, которому должна соответствовать группа или статистическое выражение.

Типы данных text, image и ntext нельзя использовать в предложении HAVING.

### **Примеры**

В следующем примере, который использует простое предложение HAVING, из таблицы

SalesOrderDetail извлекается сумма всех полей SalesOrderID, значение которых превышает \$100000.00.

```
USE AdventureWorks2008R2 ;
GO
SELECT SalesOrderID, SUM(LineTotal) AS SubTotal
FROM Sales.SalesOrderDetail
GROUP BY SalesOrderID
HAVING SUM(LineTotal) > 100000.00
ORDER BY SalesOrderID ;
```

## Предложение INTO (Transact-SQL)

Инструкция SELECT...INTO создает новую таблицу в файловой группе по умолчанию и вставляет в нее результирующие строки из запроса. Полный синтаксис SELECT см. в разделе [SELECT \(Transact-SQL\)](#).

### Синтаксис

```
[ INTO new_table ]
```

### Аргументы

new\_table

Указывает имя новой таблицы, создаваемой на основе столбцов, указанных в списке выбора, и строк, выбираемых из источника данных.

Формат аргумента new\_table определяется путем расчета выражений, указанных в списке выбора. Столбцы в таблице, указанной в аргументе new\_table, создаются в порядке, соответствующем списку выбора. Все столбцы таблицы, указанной в аргументе new\_table, получают такие же имена, значения, типы данных и свойства допустимости значений NULL, которые указаны в соответствующем выражении в списке выбора. Свойство IDENTITY столбца переносится за исключением случаев, когда наступают условия, описанные в подразделе «Примечания» раздела «Работа со столбцами идентификаторов».

Чтобы создать таблицу в другой базе данных в том же экземпляре SQL Server, укажите в аргументе new\_table полное имя в виде database.schema.table\_name.

Таблицу new\_table нельзя создать на удаленном сервере, однако ее можно заполнить из удаленного источника данных. Чтобы создать таблицу new\_table из удаленной исходной таблицы, укажите источник, задав четырехкомпонентное имя в виде linked\_server.catalog.schema.object в предложении FROM инструкции SELECT. Для указания удаленного источника данных также можно использовать функцию [OPENQUERY](#) или функцию [OPENDATASOURCE](#) в предложении FROM.

### Типы данных

Атрибут FILESTREAM не передается в новую таблицу. Объекты BLOB FILESTREAM копируются и хранятся в новой таблице как объекты BLOB типа varbinary(max). Без атрибута FILESTREAM тип данных varbinary(max) имеет ограничение в 2 ГБ. Если размер большого двоичного объекта FILESTREAM превышает это значение, происходит ошибка 7119 и инструкция прекращает работу.

При выборе существующего столбца идентификаторов в новой таблице новый столбец наследует свойство `IDENTITY`, если не выполняется ни одно из следующих условий:

- инструкция `SELECT` содержит соединение, предложение `GROUP BY` или агрегатную функцию;
- несколько инструкций `SELECT` соединены при помощи `UNION`;
- столбец идентификаторов встречается более чем один раз в списке выбора;
- столбец идентификаторов является частью выражения;
- столбец идентификаторов получен из удаленного источника данных.

Если любое из этих условий выполняется, столбец создается как `NOT NULL` и не наследует свойство `IDENTITY`. Если в новой таблице необходим столбец идентификаторов, но такой столбец недоступен или необходимо изменить начальное значение или шаг приращения по сравнению с исходным столбцом идентификаторов, определите столбец в списке выбора с помощью функции `IDENTITY`. См. подраздел «Создание столбца идентификаторов с помощью функции `IDENTITY`» далее в разделе «Примеры».

## Ограничения

В качестве новой таблицы нельзя указывать табличную переменную или возвращающий табличное значение параметр.

Инструкцию `SELECT...INTO` нельзя использовать для создания секционированной таблицы, даже если исходная таблица является секционированной. Инструкция `SELECT...INTO` не использует схему секционирования исходной таблицы. Вместо этого новая таблица создается в файловой группе по умолчанию. Для вставки строк в секционированную таблицу необходимо сначала создать секционированную таблицу, а затем использовать инструкцию `INSERT INTO...SELECT FROM`.

Инструкцию `SELECT...INTO` нельзя использовать вместе с предложением [COMPUTE](#).

Индексы, ограничения и триггеры, определенные в исходной таблице, не переносятся в новую таблицу, их также нельзя указывать в инструкции `SELECT...INTO`. Если эти объекты нужны для дальнейшей работы, их необходимо создать после выполнения инструкции `SELECT...INTO`.

Указание предложения `ORDER BY` не гарантирует, что строки будут вставлены в указанном порядке.

Если в список выбора входит разреженный столбец, свойство разреженного столбца не передается в столбец в новой таблице. Если это свойство необходимо в новой таблице, измените определение столбца после выполнения инструкции `SELECT...INTO` для включения этого свойства.

Если в список выбора входит вычисляемый столбец, соответствующий столбец новой таблицы не будет вычисляемым. Значениями нового столбца становятся значения, вычисленные при выполнении инструкции `SELECT...INTO`.



## Режим ведения журнала

Объем информации, записываемой в журнал для операции SELECT...INTO, зависит от модели восстановления, действующей для базы данных. В модели восстановления с неполным протоколированием и в простой модели восстановления минимально протоколируются массовые операции. При минимальном ведении журнала использование инструкции SELECT... INTO может оказаться более эффективным, чем создание таблицы и заполнение ее инструкцией INSERT. Дополнительные сведения см. в разделе [Операции, для которых возможно минимальное протоколирование](#).

## Разрешения

Требуется разрешение CREATE TABLE в целевой базе данных.

## Примеры

### ***А. Создание таблицы путем указания столбцов из нескольких источников***

В следующем примере таблица dbo.EmployeeAddresses создается с помощью выбора семи столбцов из различных таблиц, содержащих сведения о сотрудниках и адресах.

```
USE AdventureWorks2008R2;
GO
SELECT c.FirstName, c.LastName, e.JobTitle, a.AddressLine1, a.City,
       sp.Name AS [State/Province], a.PostalCode
INTO dbo.EmployeeAddresses
FROM Person.Person AS c
     JOIN HumanResources.Employee AS e
       ON e.BusinessEntityID = c.BusinessEntityID
     JOIN Person.BusinessEntityAddress AS bea
       ON e.BusinessEntityID = bea.BusinessEntityID
     JOIN Person.Address AS a
       ON bea.AddressID = a.AddressID
     JOIN Person.StateProvince AS sp
       ON sp.StateProvinceID = a.StateProvinceID;
GO
```

### ***Б. Вставка строк с применением минимального протоколирования***

В следующем примере создается таблица dbo.NewProducts, а затем вставляются строки из таблицы Production.Product. В примере предполагается, что для базы данных База данных AdventureWorks2008R2 выбрана модель восстановления FULL. Чтобы убедиться, что применяется минимальное протоколирование, модель восстановления базы данных База данных AdventureWorks2008R2 устанавливается в значение BULK\_LOGGED перед вставкой строк и возвращается в значение FULL после инструкции SELECT...INTO. Эта процедура обеспечивает минимальное использование журнала транзакций инструкцией SELECT...INTO и ее эффективное выполнение.

```
USE AdventureWorks2008R2;
GO
IF OBJECT_ID ('dbo.NewProducts', 'U') IS NOT NULL
    DROP TABLE dbo.NewProducts;
GO
ALTER DATABASE AdventureWorks2008R2 SET RECOVERY BULK_LOGGED;
GO

SELECT * INTO dbo.NewProducts
```

```
FROM Production.Product
WHERE ListPrice > $25
AND ListPrice < $100;
GO
ALTER DATABASE AdventureWorks2008R2 SET RECOVERY FULL;
GO
```

### ***В. Создание столбца идентификаторов с помощью функции IDENTITY***

В следующем примере используется функция IDENTITY для создания столбца идентификаторов в новой таблице Person.USAddress. Это необходимо, поскольку инструкция SELECT, которая определяет таблицу, содержит соединение, и в результате свойство IDENTITY не переносится в новую таблицу. Обратите внимание, что начальное значение и шаг приращения, заданные в функции IDENTITY, отличаются от значений в столбце AddressID исходной таблицы Person.Address.

```
USE AdventureWorks2008R2;
GO
IF OBJECT_ID ('Person.USAddress') IS NOT NULL
DROP TABLE Person.USAddress;
GO
-- Determine the IDENTITY status of the source column AddressID.
SELECT OBJECT_NAME(object_id) AS TableName, name AS column_name, is_identity,
seed_value, increment_value
FROM sys.identity_columns
WHERE name = 'AddressID';

-- Create a new table with columns from the existing table Person.Address. A new
IDENTITY
-- column is created by using the IDENTITY function.
SELECT IDENTITY (int, 100, 5) AS AddressID,
       a.AddressLine1, a.City, b.Name AS State, a.PostalCode
INTO Person.USAddress
FROM Person.Address AS a
INNER JOIN Person.StateProvince AS b ON a.StateProvinceID = b.StateProvinceID
WHERE b.CountryRegionCode = N'US';

-- Verify the IDENTITY status of the AddressID columns in both tables.
SELECT OBJECT_NAME(object_id) AS TableName, name AS column_name, is_identity,
seed_value, increment_value
FROM sys.identity_columns
WHERE name = 'AddressID';
```

### ***Г. Создание таблицы путем указания столбцов из удаленного источника данных***

В следующем примере показаны три метода создания новой таблицы на локальном сервере из удаленного источника данных. Пример начинается с создания ссылки на удаленный источник данных. Затем задается имя связанного сервера (MyLinkServer,) в предложении FROM первой инструкции SELECT...INTO и в функции OPENQUERY второй инструкции SELECT...INTO. В третьей инструкции SELECT...INTO используется функция OPENDATASOURCE, которая непосредственно задает удаленный источник данных, не указывая имя связанного сервера.

```
USE master;
GO
-- Create a link to the remote data source.
-- Specify a valid server name for @datasrc as 'server_name' or
'server_name\instance_name'.
EXEC sp_addlinkedserver @server = N'MyLinkServer',
```

```
@srvproduct = N' ',
@provider = N'SQLNCLI',
@datasrc = N'server_name',
@catalog = N'AdventureWorks2008R2';
GO
USE AdventureWorks2008R2;
GO
-- Specify the remote data source in the FROM clause using a four-part name
-- in the form linked_server.catalog.schema.object.
SELECT *
INTO dbo.Departments
FROM MyLinkServer.AdventureWorks2008R2.HumanResources.Department
GO
-- Use the OPENQUERY function to access the remote data source.
SELECT *
INTO dbo.DepartmentsUsingOpenQuery
FROM OPENQUERY(MyLinkServer, 'SELECT *
                             FROM AdventureWorks2008R2.HumanResources.Department');
GO
-- Use the OPENDATASOURCE function to specify the remote data source.
-- Specify a valid server name for Data Source using the format server_name or
server_name\instance_name.
SELECT *
INTO dbo.DepartmentsUsingOpenDataSource
FROM OPENDATASOURCE('SQLNCLI',
                    'Data Source=server_name;Integrated Security=SSPI')
    .AdventureWorks2008R2.HumanResources.Department;
GO
```

## Предложение ORDER BY (Transact-SQL)

Указывает порядок сортировки для столбцов, возвращаемых инструкцией SELECT.

Предложение ORDER BY не может применяться в представлениях, встроенных функциях, производных таблицах и вложенных запросах, если не указано предложение TOP.

### Примечание

При использовании предложения ORDER BY в определении представления, встроенной функции, производной таблице или вложенном запросе, предложение используется только для определения строк, возвращаемых предложением TOP. Предложение ORDER BY не гарантирует упорядочивания результатов при запросе этих конструкций, если оно не указано в самом запросе.

### Синтаксис

```
[ ORDER BY
    {
        order_by_expression
    [ COLLATE collation_name ]
    [ ASC | DESC ]
    } [ ,...n ]
]
```

### Аргументы

order\_by\_expression

Указывает столбец, по которому должна выполняться сортировка. Столбец сортировки может быть указан с помощью имени или псевдонима столбца или неотрицательного целого числа, представляющего позицию имени или псевдонима в списке выбора. Нельзя указывать целое число, если аргумент `order_by_expression` присутствует в ранжирующей функции. Столбец сортировки может включать выражение, но если база данных находится в режиме совместимости SQL Server (90), то выражение не может быть преобразовано в константу. Имена и псевдонимы столбцов могут быть дополнены именем таблицы или представления. В SQL Server уточненные имена и псевдонимы столбцов связываются со столбцами, перечисленными в предложении FROM. Если в выражении `order_by_expression` отсутствует квалификатор, то значение должно быть уникальным во всех столбцах, перечисленных в инструкции SELECT.

Можно указать несколько столбцов сортировки. Последовательность столбцов сортировки в предложении ORDER BY определяет организацию упорядоченного результирующего набора.

В предложение ORDER BY могут входить элементы, которых нет в списке выборки. Однако если указана конструкция SELECT DISTINCT, или инструкция содержит предложение GROUP BY, или если инструкция SELECT содержит оператор UNION, то столбцы сортировки должны присутствовать в списке выборки.

Кроме того, если в инструкцию SELECT входит оператор UNION, то имена и псевдонимы столбцов должны быть из числа уточненных в первом списке выбора.

### Примечание

Столбцы типа ntext, text, image или xml не могут быть использованы в предложении ORDER BY.

### COLLATE {collation\_name}

Указывает, что операция ORDER BY должна выполняться в соответствии с параметрами сортировки, указанными в аргументе collation\_name, но не в соответствии с параметрами сортировки столбца, определенных в таблице или представлении. Значение collation\_name может быть именем параметров сортировки Windows или именем параметров сортировки SQL. Дополнительные сведения см. в разделах [Настройка параметров сортировки в программе установки](#) и [Использование параметров сортировки SQL Server](#). Аргумент COLLATE применяется только к столбцам данных типа char, varchar, nchar и nvarchar.

### ASC

Указывает, что значения в указанном столбце должны сортироваться по возрастанию, от меньших значений к большим значениям. Сортировка по умолчанию — ASC.

### DESC

Указывает, что значения в указанном столбце должны сортироваться по убыванию, от больших значений к меньшим.

## Замечания

Значения NULL рассматриваются как минимально возможные значения.

Число элементов в предложении ORDER BY не ограничивается. Однако существует ограничение в 8 060 байт для размера строки промежуточных рабочих таблиц, необходимых для операций сортировки. Это ограничивает общий размер столбцов, указываемый в предложении ORDER BY.

Использование предложения ORDER BY с инструкцией SELECT...INTO для вставки строк из другого источника, не гарантирует вставку строк в указанном порядке.

## Примеры

В следующих примерах демонстрируется упорядочивание результирующего набора.

Упорядочивание по числовому столбцу ProductID. По умолчанию задан порядок по возрастанию.

```
USE AdventureWorks2008R2;  
GO  
SELECT ProductID, Name FROM Production.Product  
WHERE Name LIKE 'Lock Washer%'  
ORDER BY ProductID ;
```

Упорядочивание по числовому столбцу ProductID в порядке по убыванию.

```
SELECT ProductID, Name FROM Production.Product  
WHERE Name LIKE 'Lock Washer%'  
ORDER BY ProductID DESC;
```

Упорядочивание по столбцу Name. Обратите внимание, что символьные значения сортируются в алфавитном порядке, а не в числовом. Иными словами, отсортированное значение 10 находится перед 2.

```
SELECT ProductID, Name FROM Production.Product  
WHERE Name LIKE 'Lock Washer%'  
ORDER BY Name ASC ;
```

Упорядочивание по двум столбцам. В этом запрос сначала осуществляется сортировка по возрастанию значений в столбце FirstName, а затем — по убыванию значений в столбце LastName.

```
SELECT LastName, FirstName FROM Person.Person  
WHERE LastName LIKE 'R%'  
ORDER BY FirstName ASC, LastName DESC ;
```

## Предложение OVER (Transact-SQL)

Определяет секционирование и упорядочение набора строк до применения соответствующей оконной функции.

Ранжирующие оконные функции

Статистические оконные функции. Дополнительные сведения см. в разделе

## Статистические функции (Transact-SQL).

### Синтаксис

#### Ranking Window Functions

```
< OVER_CLAUSE > :: =  
    OVER ( [ PARTITION BY value_expression , ... [ n ] ]  
          <ORDER BY Clause> )
```

#### Aggregate Window Functions

```
< OVER_CLAUSE > :: =  
    OVER ( [ PARTITION BY value_expression , ... [ n ] ] )
```

### Аргументы

#### PARTITION BY

Разделяет результирующий набор на секции. Оконная функция применяется к каждой секции отдельно, и вычисление начинается заново для каждой секции.

#### value\_expression

Указывает столбец, по которому секционируется набор строк, произведенный соответствующим предложением FROM. Аргумент value\_expression может ссылаться только на столбцы, доступные через предложение FROM. Выражение value\_expression не может ссылаться на выражения или псевдонимы в списке выбора. value\_expression может быть выражением столбца, скалярным вложенным запросом, скалярной функцией или пользовательской переменной.

#### <Предложение ORDER BY>

Задаёт порядок для ранжирующей оконной функции. Дополнительные сведения см. в разделе [Предложение ORDER BY \(Transact-SQL\)](#).

### Важно!

Если <предложение ORDER BY> используется в контексте ранжирующей оконной функции, оно может ссылаться только на столбцы, доступные через предложение FROM. Указывать положение имени или псевдонима столбца в списке выборки с помощью целого числа нельзя. <Предложение ORDER BY> не может использоваться со статистическими оконными функциями.

### Замечания

Оконные функции определены в стандарте ISO SQL. В SQL Server предоставляются ранжирующие и статистические оконные функции. Окно — это набор строк, определяемый пользователем. Оконная функция вычисляет значение для каждой строки в результирующем наборе, полученном из окна.

В одном запросе с одним предложением FROM может использоваться несколько статистических или ранжирующих оконных функций. Однако предложение OVER для каждой функции может использовать свое секционирование и упорядочение.

Предложение OVER не может использоваться со статистической функцией CHECKSUM.

## Примеры

### ***А. Использование предложения OVER с функцией ROW\_NUMBER***

Каждая из ранжирующих функций ROW\_NUMBER, DENSE\_RANK, RANK и NTILE использует предложение OVER. Следующий пример демонстрирует использование предложения OVER с функцией ROW\_NUMBER.

```
USE AdventureWorks2008R2;
GO
SELECT p.FirstName, p.LastName
      ,ROW_NUMBER() OVER(ORDER BY SalesYTD DESC) AS 'Row Number'
      ,s.SalesYTD, a.PostalCode
FROM Sales.SalesPerson s
     INNER JOIN Person.Person p
          ON s.BusinessEntityID = p.BusinessEntityID
     INNER JOIN Person.Address a
          ON a.AddressID = p.BusinessEntityID
WHERE TerritoryID IS NOT NULL
     AND SalesYTD <> 0;
GO
```

### ***Б. Использование предложения OVER с агрегатными функциями***

Следующие примеры демонстрируют использование предложения OVER с агрегатными функциями. В данном примере использование предложения OVER является более эффективным, чем использование вложенных запросов.

```
USE AdventureWorks2008R2;
GO
SELECT SalesOrderID, ProductID, OrderQty
      ,SUM(OrderQty) OVER(PARTITION BY SalesOrderID) AS 'Total'
      ,AVG(OrderQty) OVER(PARTITION BY SalesOrderID) AS 'Avg'
      ,COUNT(OrderQty) OVER(PARTITION BY SalesOrderID) AS 'Count'
      ,MIN(OrderQty) OVER(PARTITION BY SalesOrderID) AS 'Min'
      ,MAX(OrderQty) OVER(PARTITION BY SalesOrderID) AS 'Max'
FROM Sales.SalesOrderDetail
WHERE SalesOrderID IN(43659,43664);
GO
```

Ниже приводится результирующий набор.

| SalesOrderID | ProductID | OrderQty | Total | Avg | Count | Min | Max |
|--------------|-----------|----------|-------|-----|-------|-----|-----|
| 43659        | 776       | 1        | 26    | 2   | 12    | 1   | 6   |
| 43659        | 777       | 3        | 26    | 2   | 12    | 1   | 6   |
| 43659        | 778       | 1        | 26    | 2   | 12    | 1   | 6   |
| 43659        | 771       | 1        | 26    | 2   | 12    | 1   | 6   |
| 43659        | 772       | 1        | 26    | 2   | 12    | 1   | 6   |
| 43659        | 773       | 2        | 26    | 2   | 12    | 1   | 6   |

|       |     |   |    |   |    |   |   |
|-------|-----|---|----|---|----|---|---|
| 43659 | 774 | 1 | 26 | 2 | 12 | 1 | 6 |
| 43659 | 714 | 3 | 26 | 2 | 12 | 1 | 6 |
| 43659 | 716 | 1 | 26 | 2 | 12 | 1 | 6 |
| 43659 | 709 | 6 | 26 | 2 | 12 | 1 | 6 |
| 43659 | 712 | 2 | 26 | 2 | 12 | 1 | 6 |
| 43659 | 711 | 4 | 26 | 2 | 12 | 1 | 6 |
| 43664 | 772 | 1 | 14 | 1 | 8  | 1 | 4 |
| 43664 | 775 | 4 | 14 | 1 | 8  | 1 | 4 |
| 43664 | 714 | 1 | 14 | 1 | 8  | 1 | 4 |
| 43664 | 716 | 1 | 14 | 1 | 8  | 1 | 4 |
| 43664 | 777 | 2 | 14 | 1 | 8  | 1 | 4 |
| 43664 | 771 | 3 | 14 | 1 | 8  | 1 | 4 |
| 43664 | 773 | 1 | 14 | 1 | 8  | 1 | 4 |
| 43664 | 778 | 1 | 14 | 1 | 8  | 1 | 4 |

Следующий пример демонстрирует использование предложения OVER со статистической функцией в вычисляемом значении.

```
USE AdventureWorks2008R2;
GO
SELECT SalesOrderID, ProductID, OrderQty
      ,SUM(OrderQty) OVER(PARTITION BY SalesOrderID) AS 'Total'
      ,CAST(1. * OrderQty / SUM(OrderQty) OVER(PARTITION BY SalesOrderID)
          *100 AS DECIMAL(5,2))AS 'Percent by ProductID'
FROM Sales.SalesOrderDetail
WHERE SalesOrderID IN(43659,43664);
GO
```

Ниже приводится результирующий набор. Обратите внимание, что статистические функции вычисляются в столбце SalesOrderID, а столбец Percent by ProductID вычисляется для каждой строки каждого SalesOrderID.

| SalesOrderID | ProductID | OrderQty | Total | Percent by ProductID |
|--------------|-----------|----------|-------|----------------------|
|--------------|-----------|----------|-------|----------------------|



|       |     |   |    |       |
|-------|-----|---|----|-------|
| 43659 | 776 | 1 | 26 | 3.85  |
| 43659 | 777 | 3 | 26 | 11.54 |
| 43659 | 778 | 1 | 26 | 3.85  |
| 43659 | 771 | 1 | 26 | 3.85  |
| 43659 | 772 | 1 | 26 | 3.85  |
| 43659 | 773 | 2 | 26 | 7.69  |
| 43659 | 774 | 1 | 26 | 3.85  |
| 43659 | 714 | 3 | 26 | 11.54 |
| 43659 | 716 | 1 | 26 | 3.85  |
| 43659 | 709 | 6 | 26 | 23.08 |
| 43659 | 712 | 2 | 26 | 7.69  |
| 43659 | 711 | 4 | 26 | 15.38 |
| 43664 | 772 | 1 | 14 | 7.14  |
| 43664 | 775 | 4 | 14 | 28.57 |
| 43664 | 714 | 1 | 14 | 7.14  |
| 43664 | 716 | 1 | 14 | 7.14  |
| 43664 | 777 | 2 | 14 | 14.29 |
| 43664 | 771 | 3 | 14 | 21.43 |
| 43664 | 773 | 1 | 14 | 7.14  |
| 43664 | 778 | 1 | 14 | 7.14  |

## sys.tables (Transact-SQL)

Возвращает по строке для каждого объекта таблицы, на данный момент только с типом sys.objects.type = U.

| Имя столбца           | Тип данных | Описание                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
|-----------------------|------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <наследуемые столбцы> |            | Список столбцов, наследуемых данным представлением, см. в разделе <a href="#">sys.objects (Transact-SQL)</a> .                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| lob_data_space_id     | int        | <p>Ненулевое значение — это идентификатор пространства данных (файловой группы или схемы секционирования), содержащего данные типов text, ntext и image для данной таблицы.</p> <p>0 = Таблица не содержит данных типов text, ntext или image.</p> <p>Идентификатор пространства данных для файловой группы FILESTREAM или схемы секционирования, состоящей из файловых групп FILESTREAM.</p> <p>Чтобы получить имя файловой группы FILESTREAM, необходимо выполнить запрос SELECT FILEGROUP_NAME (filestream_data_space_id) FROM sys.tables.</p> <p>sys.tables можно объединить со следующими представлениями по условию filestream_data_space_id = data_space_id.</p> <ul style="list-style-type: none"> <li>• sys.filegroups</li> <li>• sys.partition_schemes</li> <li>• sys.indexes</li> <li>• sys.allocation_units</li> <li>• sys.fulltext_catalogs</li> <li>• sys.data_spaces</li> <li>• sys.destination_data_spaces</li> <li>• sys.master_files</li> <li>• sys.database_files</li> <li>• backupfilegroup (соединенная по столбцу filegroup_id)</li> </ul> |
| max_column_id_used    | int        | Максимальный идентификатор столбца, когда-либо использованный в таблице.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |

|                               |              |                                                                                                                                                                                                                                           |
|-------------------------------|--------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| lock_on_bulk_load             | bit          | Таблица заблокирована при массовой загрузке. Дополнительные сведения см. в разделе <a href="#">sp_tableoption (Transact-SQL)</a> .                                                                                                        |
| uses_ansi_nulls               | bit          | Таблица была создана при установленном параметре SET ANSI_NULLS = ON.                                                                                                                                                                     |
| is_replicated                 | bit          | 1 = Таблица опубликована путем репликации моментальных снимков или транзакций.                                                                                                                                                            |
| has_replication_filter        | bit          | 1 = Для таблицы имеется фильтр репликации.                                                                                                                                                                                                |
| is_merge_published            | bit          | 1 = Таблица опубликована путем репликации слиянием.                                                                                                                                                                                       |
| is_sync_tran_subscribed       | bit          | 1 = Таблица используется в немедленно обновляемой подписке.                                                                                                                                                                               |
| has_unchecked_assembly_data   | bit          | 1 = Таблица содержит сохраняемые данные, зависящие от сборки, определение которой изменилось за время последней операции ALTER ASSEMBLY. Значение будет сброшено на 0 после следующей успешной операции DBCC CHECKDB или DBCC CHECKTABLE. |
| text_in_row_limit             | int          | Максимальное число байтов, разрешенное для текста в строке.<br><br>0 = Параметр текста в строке не установлен. Дополнительные сведения см. в разделе <a href="#">sp_tableoption (Transact-SQL)</a> .                                      |
| large_value_type_s_out_of_row | bit          | 1 = Типы больших значений хранятся вне строк. Дополнительные сведения см. в разделе <a href="#">sp_tableoption (Transact-SQL)</a> .                                                                                                       |
| is_tracked_by_cdc             | bit          | 1 = в таблице включена система отслеживания измененных данных. Дополнительные сведения см. в разделе <a href="#">sys.sp_cdc_enable_table (Transact-SQL)</a> .                                                                             |
| lock_escalation               | tinyint      | Значение параметра LOCK_ESCALATION для таблицы.<br><br>0 = TABLE<br><br>1 = DISABLE<br><br>2 = AUTO                                                                                                                                       |
| lock_escalation_desc          | nvarchar(60) | Текстовое описание параметра lock_escalation для таблицы. Возможные значения перечислены ниже. TABLE, AUTO и                                                                                                                              |

DISABLE.

### Разрешения

В SQL Server 2005 и более поздних версиях видимость метаданных в представлениях каталогов ограничивается защищаемыми объектами, которыми пользователь владеет или на которые ему были предоставлены разрешения. Дополнительные сведения см. в разделе [Настройка видимости метаданных](#).

## sys.columns (Transact-SQL)

Возвращает строку для каждого столбца объекта, имеющего столбцы, например представления или таблицы. Далее следует список типов объектов, имеющих столбцы:

- возвращающие табличное значение функции сборки (FT);
- встроенные возвращающие табличное значение функции SQL (IF);
- внутренние таблицы (IT);
- системные таблицы (S);
- возвращающие табличное значение функции SQL (TF);
- пользовательские таблицы (U);
- представления (V).

| Имя столбца    | Тип данных | Описание                                                                                                         |
|----------------|------------|------------------------------------------------------------------------------------------------------------------|
| object_id      | int        | Идентификатор объекта, которому принадлежит этот столбец.                                                        |
| name           | sysname    | Имя столбца. Уникально в рамках объекта.                                                                         |
| column_id      | int        | Идентификатор столбца. Уникален в рамках объекта.                                                                |
|                |            | Идентификаторы столбцов могут не быть последовательными.                                                         |
| system_type_id | tinyint    | Идентификатор системного типа столбца.                                                                           |
| user_type_id   | int        | Идентификатор типа столбца, определенного пользователем.                                                         |
|                |            | Чтобы вернуть имя типа, создайте в этом столбце соединение с представлением каталога <a href="#">sys.types</a> . |
| max_length     | smallint   | Максимальная длина столбца (в байтах).                                                                           |
|                |            | -1 = тип данных столбца — varchar(max), nvarchar(max),                                                           |

|                       |         |                                                                                                                                                                            |
|-----------------------|---------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|                       |         | varbinary(max) или xml.                                                                                                                                                    |
|                       |         | Для столбцов типа text значение max_length должно быть равно 16 или равно значению аргумента «text in row», установленному процедурой sp_tableoption.                      |
| precision             | tinyint | Точность чисел в столбце, если он является цифровым; в противном случае — 0.                                                                                               |
| scale                 | tinyint | Масштаб значений столбца в случае числового выражения; в противном случае — 0.                                                                                             |
| collation_name        | sysname | Имя параметров сортировки столбца, если он является символьным; в противном случае — значение NULL.                                                                        |
| is_nullable           | bit     | 1 = ячейки в столбце могут принимать значения NULL.                                                                                                                        |
| is_ansi_padded        | bit     | 1 = столбец использует поведение ANSI_PADDING ON, если принадлежит типам character, binary или variant.<br>0 = столбец не принадлежит типам character, binary или variant. |
| is_rowguidcol         | bit     | 1 = столбец является объявленным ROWGUIDCOL.                                                                                                                               |
| is_identity           | bit     | 1 = столбец содержит значения identity.                                                                                                                                    |
| is_computed           | bit     | 1 = столбец является вычисляемым.                                                                                                                                          |
| is_filestream         | bit     | 1 — столбец является столбцом FILESTREAM.                                                                                                                                  |
| is_replicated         | bit     | 1 = столбец реплицируется.                                                                                                                                                 |
| is_non_sql_subscribed | bit     | 1 = у столбца есть подписчик, отличный от подписчика SQL Server.                                                                                                           |
| is_merge_published    | bit     | 1 = столбец публикуется слиянием.                                                                                                                                          |
| is_dts_replicated     | bit     | 1 = столбец реплицируется с использованием служб SSIS.                                                                                                                     |
| is_xml_document       | bit     | 1 = содержимое является полным XML-документом.<br>0 = содержимое является фрагментом документа, или тип данных столбца не является xml.                                    |
| xml_collection_id     | int     | Ненулевое значение, если тип данных столбца является xml, а XML типизирован. Значением будет идентификатор коллекции,                                                      |

|                   |     |                                                                                                                                                                                                                                                                                                                   |
|-------------------|-----|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|                   |     | содержащей подтвержденное пространство имен XML-схемы для столбца.                                                                                                                                                                                                                                                |
|                   |     | 0 = нет коллекции XML-схемы.                                                                                                                                                                                                                                                                                      |
| default_object_id | int | Идентификатор объекта по умолчанию независит от того, является ли объект автономным объектом <a href="#">sys.sp_bindefault</a> или встроенным ограничением DEFAULT уровня столбца. Столбец parent_object_id встроенного объекта «значение по умолчанию» уровня столбца представляет собой ссылку на саму таблицу. |
|                   |     | 0 = нет значения по умолчанию.                                                                                                                                                                                                                                                                                    |
| rule_object_id    | int | Идентификатор изолированного правила, привязанного к столбцу с помощью процедуры sys.sp_bindrule.                                                                                                                                                                                                                 |
|                   |     | 0 = автономное правило отсутствует. Сведения об ограничениях инструкции CHECK на уровне столбцов см. в разделе <a href="#">sys.check_constraints (Transact-SQL)</a> .                                                                                                                                             |
| is_sparse         | bit | 1 = столбец является разреженным. Дополнительные сведения см. в разделе <a href="#">Использование разреженных столбцов</a> .                                                                                                                                                                                      |
| is_column_set     | bit | 1 = столбец является набором столбцов. Дополнительные сведения см. в разделе <a href="#">Использование разреженных столбцов</a> .                                                                                                                                                                                 |

### Разрешения

В SQL Server 2005 и более поздних версиях видимость метаданных в представлениях каталогов ограничивается защищаемыми объектами, которыми пользователь владеет или на которые ему были предоставлены разрешения. Дополнительные сведения см. в разделе [Настройка видимости метаданных](#).

## COLUMNPROPERTY (Transact-SQL)

Возвращает сведения о столбце или о параметре.

### Синтаксис

COLUMNPROPERTY ( id , column , property )

### Аргументы

id

[Выражение](#), которое содержит идентификатор таблицы или процедуры.

column

Выражение, которое содержит имя столбца или параметра.

property

Выражение, которое содержит информацию, возвращаемую для аргумента `id`, и может быть одним из следующих значений.

| Значение                  | Описание                                                                                                                             | Возвращаемое значение                                                                                                               |
|---------------------------|--------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------|
| <b>AllowsNull</b>         | Разрешение использовать NULL.                                                                                                        | 1 = TRUE;                                                                                                                           |
|                           |                                                                                                                                      | 0 = FALSE;                                                                                                                          |
|                           |                                                                                                                                      | NULL = Введенные значения недопустимы.                                                                                              |
| <b>ColumnId</b>           | Значение идентификатора столбца, соответствующего <code>sys.columns.column_id</code> .                                               | Идентификатор столбца                                                                                                               |
|                           |                                                                                                                                      | <b>Примечание</b><br>При запросе множества столбцов могут появиться пропуски в последовательности значений идентификаторов столбца. |
| <b>FullTextTypeColumn</b> | TYPE COLUMN в таблице, которая содержит информацию <code>column</code> о типе документа.                                             | Идентификатор полнотекстового TYPE COLUMN для столбцов, переданных вторым параметром этого свойства.                                |
|                           |                                                                                                                                      |                                                                                                                                     |
| <b>IsComputed</b>         | Столбец является вычисляемым столбцом.                                                                                               | 1 = TRUE;                                                                                                                           |
|                           |                                                                                                                                      | 0 = FALSE;                                                                                                                          |
|                           |                                                                                                                                      | NULL = Введенные значения недопустимы.                                                                                              |
| <b>IsCursorType</b>       | Параметр процедуры имеет тип CURSOR.                                                                                                 | 1 = TRUE;                                                                                                                           |
|                           |                                                                                                                                      | 0 = FALSE;                                                                                                                          |
|                           |                                                                                                                                      | NULL = Введенные значения недопустимы.                                                                                              |
| <b>IsDeterministic</b>    | Столбцы являются детерминированными (предсказуемыми). Это свойство применимо только к вычисляемым столбцам и столбцам представлений. | 1 = TRUE;                                                                                                                           |
|                           |                                                                                                                                      | 0 = FALSE;                                                                                                                          |
|                           |                                                                                                                                      | NULL = Введенные значения недопустимы.                                                                                              |

|                          |                                                                              |                                                                                 |
|--------------------------|------------------------------------------------------------------------------|---------------------------------------------------------------------------------|
|                          |                                                                              | Невычисляемый столбец или не столбец представлений.                             |
|                          |                                                                              | 1 = TRUE;                                                                       |
| <b>IsFulltextIndexed</b> | Столбцы отмечены для полнотекстовой индексации.                              | 0 = FALSE;<br><br>NULL = Введенные значения недопустимы.                        |
|                          |                                                                              | 1 = TRUE;                                                                       |
| <b>IsIdentity</b>        | Столбец использует свойство IDENTITY.                                        | 0 = FALSE;<br><br>NULL = Введенные значения недопустимы.                        |
|                          |                                                                              | 1 = TRUE                                                                        |
| <b>IsIdNotForRepl</b>    | Столбец проверяет настройку IDENTITY_INSERT.                                 | 0 = FALSE<br><br>NULL = Введенные значения недопустимы.                         |
|                          |                                                                              | 1 = TRUE;                                                                       |
| <b>IsIndexable</b>       | Столбцы не могут быть индексированы.                                         | 0 = FALSE;<br><br>NULL = Введенные значения недопустимы.                        |
|                          |                                                                              | 1 = TRUE;                                                                       |
| <b>IsOutParam</b>        | Параметр процедуры является выходным параметром.                             | 0 = FALSE NULL = Введенные значения недопустимы.                                |
|                          |                                                                              | 1 = TRUE;                                                                       |
| <b>IsPrecise</b>         | Точный столбец. Это свойство применимо только к детерминированным столбцам.  | 0 = FALSE NULL = Введенные значения недопустимы.<br>Недетерминированный столбец |
| <b>IsRowGuidCol</b>      | Столбец имеет тип данных uniqueidentifier и определяется вместе со свойством | 1 = TRUE;<br><br>0 = FALSE;                                                     |



|                         | ROWGUIDCOL.                                                                                                                                                                                                                        | NULL = Введенные значения недопустимы.                                                                                      |
|-------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------|
| <b>IsSystemVerified</b> | Свойства детерминированности и точности столбца могут быть проверены с помощью Database Engine. Это свойство применимо только к вычисляемым столбцам и столбцам представлений.                                                     | 1 = TRUE;<br>0 = FALSE;<br><br>NULL = Введенные значения недопустимы.                                                       |
| <b>IsXmlIndexable</b>   | XML-столбец может быть использован в XML-индексе.                                                                                                                                                                                  | 1 = TRUE;<br>0 = FALSE;<br><br>NULL = Введенные значения недопустимы.                                                       |
| <b>Precision</b>        | Длина для типа данных в столбце или для параметра.                                                                                                                                                                                 | Длина заданного типа данных столбца<br><br>-1 = xml или типы больших значений<br><br>NULL = Введенные значения недопустимы. |
| <b>Scale</b>            | Масштаб для типа данных в столбце или для параметра.                                                                                                                                                                               | Масштаб<br><br>NULL = Введенные значения недопустимы.                                                                       |
| <b>SystemDataAccess</b> | Столбец, полученный из функции, которая получает данные в системных каталогах или виртуальных системных таблицах SQL Server. Это свойство применимо только к вычисляемым столбцам и столбцам представлений.                        | 1 = TRUE (обозначает доступ только для чтения)<br>0 = FALSE;<br><br>NULL = Введенные значения недопустимы.                  |
| <b>UserDataAccess</b>   | Столбец, полученный из функции, которая получает данные в пользовательских таблицах, включая таблицы видов и временные таблицы, хранится в локальном экземпляре SQL Server. Это свойство применимо только к вычисляемым столбцам и | 1 = TRUE (обозначает доступ только для чтения)<br>0 = FALSE;<br><br>NULL = Введенные значения недопустимы.                  |

столбцам представлений.

|                     |                                                                                                                                                       |                                                               |
|---------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------|
| <b>UsesAnsiTrim</b> | ANSI_PADDING был установлен как ON, если таблица была создана впервые. Это свойство применимо только к столбцам или параметрам типа char или varchar. | 1= TRUE<br>0= FALSE<br>NULL = Введенные значения недопустимы. |
| <b>IsSparse</b>     | Столбец является разреженным столбцом. Дополнительные сведения см. в разделе <a href="#">Использование разреженных столбцов</a> .                     | 1= TRUE<br>0= FALSE<br>NULL = Введенные значения недопустимы. |
| <b>IsColumnSet</b>  | Столбец представляет собой набор столбцов. Дополнительные сведения см. в разделе <a href="#">Использование наборов столбцов</a> .                     | 1= TRUE<br>0= FALSE<br>NULL = Введенные значения недопустимы. |

## Типы возвращаемых данных

int

## Исключения

Возвращает значение NULL в случае ошибки или отсутствия у участника разрешения на просмотр объекта.

В SQL Server 2008 пользователь может просматривать только метаданные защищаемых объектов, которыми он владеет или на которые ему были предоставлены разрешения. Это означает, что встроенные функции, создающие метаданные, такие как COLUMNPROPERTY, могут вернуть значение NULL в случае, если пользователь не имеет разрешений на объект. Дополнительные сведения см. в разделах [Настройка видимости метаданных](#) и [Устранение неполадок, связанных с видимостью метаданных](#).

## Замечания

Если проверяется детерминистическое свойство столбца, сначала нужно проверить, является ли столбец вычисляемым. **IsDeterministic** возвращает NULL для невычисляемых столбцов. Вычисляемые столбцы могут быть определены как индексированные столбцы.

## Примеры

Следующий пример возвращает длину столбца LastName.

```
USE AdventureWorks2008R2;
GO
```

```
SELECT COLUMNPROPERTY( OBJECT_ID('Person.Person'), 'LastName', 'PRECISION')AS  
'Column Length';  
GO
```

Ниже приводится результирующий набор.

```
Column Length
```

```
-----
```

```
50
```

```
(1 row(s) affected)
```

## sys.computed\_columns (Transact-SQL)

Содержит одну строку для каждого столбца, найденного в вычисляемом столбце `sys.columns`.

| Имя столбца             | Тип данных    | Описание                                                                                                                                                                                                                                                                                                           |
|-------------------------|---------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <Наследуемые столбцы>   |               | Список столбцов, наследуемых этим представлением, см. в разделе <a href="#">sys.columns (Transact-SQL)</a> .<br><br>Текст на языке SQL, определяющий этот вычисляемый столбец.                                                                                                                                     |
| definition              | nvarchar(max) | SQL Server 2008 и SQL Server 2005 отличаются от SQL Server 2000 способом расшифровки и хранения выражений SQL в метаданных каталога. Семантика расшифрованных выражений соответствует исходному тексту, однако правильность синтаксиса не гарантируется. Например, из декодированного выражения удаляются пробелы. |
| uses_database_collation | bit           | 1 = правильное определение столбца зависит от параметров сортировки по умолчанию для базы данных; в противном случае 0. Такая зависимость предотвращает изменение сопоставления по умолчанию для базы данных.                                                                                                      |
| is_persisted            | bit           | Вычисляемый столбец сохраняется.                                                                                                                                                                                                                                                                                   |
| is_computed             | bit           | Вычисляемый столбец вычисляется. Значение всегда равно 1.                                                                                                                                                                                                                                                          |
| is_sparse               | bit           | 1 = столбец является разреженным. Дополнительные сведения см. в разделе <a href="#">Использование разреженных столбцов</a> .                                                                                                                                                                                       |
| is_column_set           | bit           | 1 = столбец является набором столбцов. Дополнительные сведения см. в разделе <a href="#">Использование наборов столбцов</a> .                                                                                                                                                                                      |

## Разрешения

В SQL Server 2005 и более поздних версиях видимость метаданных в представлениях каталогов ограничивается защищаемыми объектами, которыми пользователь владеет или на которые ему были предоставлены разрешения. Дополнительные сведения см. в разделе [Настройка видимости метаданных](#).

## Представление каталога sys.sql\_expression\_dependencies (Transact-SQL)

Содержит по одной строке для каждой именованной зависимости определяемой пользователем сущности в текущей базе данных. Зависимость между двумя сущностями создается, когда имя некоторой сущности, называемой упоминаемой, встречается в составе постоянного выражения языка SQL другой сущности, называемой ссылающейся. Например, если на таблицу ссылается определение представления, это представление, как ссылающаяся сущность, зависит от таблицы или упоминаемой сущности. При удалении таблицы представление становится непригодным для использования.

Это представление каталога можно использовать для получения сведений о зависимостях по следующим сущностям:

- привязанные к схеме сущности;
- сущности без привязки к схеме;
- межбазовые и межсерверные сущности. Выводятся имена сущностей без идентификаторов;
- зависимости на уровне столбцов в сущностях, привязанных к схеме. Зависимости на уровне столбцов для объектов, не привязанных к схеме, можно просмотреть в динамическом административном представлении [sys.dm\\_sql\\_referenced\\_entities](#);
- триггеры DDL на уровне сервера в контексте базы данных master.

| Имя столбца          | Тип данных | Описание                                                                                                                |
|----------------------|------------|-------------------------------------------------------------------------------------------------------------------------|
| referencing_id       | int        | Идентификатор ссылающейся сущности. Не допускает значение NULL.                                                         |
| referencing_minor_id | int        | Идентификатор столбца, если ссылающаяся сущность является столбцом; в противном случае — 0. Не допускает значения NULL. |
| referencing_class    | tinyint    | Класс ссылающейся сущности.                                                                                             |
| ass                  |            | 1 = Объект или столбец                                                                                                  |

|                           |              |                                                                                                     |
|---------------------------|--------------|-----------------------------------------------------------------------------------------------------|
|                           |              | 12 = Триггер DDL базы данных                                                                        |
|                           |              | 13 = Серверный триггер DDL                                                                          |
|                           |              | Не допускает значение NULL.                                                                         |
|                           |              | Описание класса ссылающейся сущности.                                                               |
|                           |              | OBJECT_OR_COLUMN                                                                                    |
| referencing_class_desc    | nvarchar(60) | DATABASE_DDL_TRIGGER                                                                                |
|                           |              | SERVER_DDL_TRIGGER                                                                                  |
|                           |              | Не допускает значение NULL.                                                                         |
| is_schema_bound_reference | bit          | 1 = Упоминаемая сущность привязана к схеме.                                                         |
|                           |              | 0 = Упоминаемая сущность не привязана к схеме.                                                      |
|                           |              | Не допускает значение NULL.                                                                         |
|                           |              | Класс упоминаемой сущности.                                                                         |
|                           |              | 1 = Объект или столбец                                                                              |
| referenced_class          | tinyint      | 6 = Тип                                                                                             |
|                           |              | 10 = Коллекция XML-схем                                                                             |
|                           |              | 21 = Функция секционирования                                                                        |
|                           |              | Не допускает значение NULL.                                                                         |
|                           |              | Описание класса упоминаемой сущности.                                                               |
|                           |              | OBJECT_OR_COLUMN                                                                                    |
| referenced_class_desc     | nvarchar(60) | TYPE                                                                                                |
|                           |              | XML_SCHEMA_COLLECTION                                                                               |
|                           |              | PARTITION_FUNCTION                                                                                  |
|                           |              | Не допускает значение NULL.                                                                         |
| referenced_server_name    | sysname      | Имя сервера упоминаемой сущности.                                                                   |
|                           |              | Этот столбец заполняется для межсерверных зависимостей, которые создаются путем задания допустимого |

четырёхкомпонентного имени. Сведения о многокомпонентных именах см. в разделе [Синтаксические обозначения в Transact-SQL \(Transact-SQL\)](#).

Значение NULL для не привязанных к схеме сущностей, ссылка на которые осуществляется без указания четырёхкомпонентного имени.

Имеет значение NULL для привязанных к схеме сущностей, поскольку они должны находиться в одной базе данных, и поэтому их можно определить только с использованием только двухкомпонентного имени (схема.объект).

Имя базы данных упоминаемой сущности.

Этот столбец заполняется для межбазовых и межсерверных ссылок, которые задаются путем указания допустимого трехкомпонентного или четырехкомпонентного имени.

referenced\_database\_name sysname

Имеет значение NULL для не привязанных к схеме ссылок, задаваемых с помощью однокомпонентного или двухкомпонентного имени.

Имеет значение NULL для привязанных к схеме сущностей, поскольку они должны находиться в одной базе данных, и поэтому их можно определить только с использованием только двухкомпонентного имени (схема.объект).

Схема, которой принадлежит упоминаемая сущность.

referenced\_schema\_name sysname

Имеет значение NULL для не привязанных к схеме ссылок, в которых сущность упоминается без указания имени схемы.

Никогда не имеет значение NULL для привязанных к схеме ссылок, поскольку привязанные к схеме сущности должны определяться двухкомпонентным именем и ссылаться с помощью двухкомпонентных ссылок.

referenced\_entity\_name sysname

Имя упоминаемой сущности. Не допускает значение NULL.

referenced\_id int

Идентификатор упоминаемой сущности.

Всегда значение NULL для межсерверных и межбазовых ссылок.

Имеет значение NULL для ссылок в пределах базы данных, когда не удастся определить идентификатор. Для ссылок, не привязанных к схеме, идентификатор не удастся разрешить в следующих случаях.

- Упоминаемая сущность не существует в базе данных.
- Схема упоминаемой сущности зависит от схемы участника и разрешается во время выполнения. В этом случае параметр `is_caller_dependent` устанавливается в значение 1.

Никогда не принимает значение NULL для привязанных к схеме ссылок.

Идентификатор ссылочного столбца в случае, если ссылающейся сущностью является столбец; в противном случае — 0. Не допускает значения NULL.

`referenced_minor_id` int

Упоминаемая сущность представляет собой столбец, если в ссылающейся сущности столбец определяется по имени или если в инструкции `SELECT *` используется родительская сущность.

Указывает, что привязка к схеме для упоминаемой сущности происходит во время выполнения, и поэтому разрешение идентификатора сущности зависит от схемы ссылающейся сущности. Это происходит, если упоминаемая сущность является хранимой процедурой, расширенной хранимой процедурой или определяемой пользователем функцией, не привязанной к схеме, вызываемой в инструкции `EXECUTE`.

`is_caller_dependent` bit

1 = Упоминаемая сущность зависит от ссылающейся и разрешается во время выполнения. В этом случае параметр `referenced_id` принимает значение NULL.

0 = Идентификатор упоминаемой сущности не зависит от вызывающего объекта.

Всегда имеет значение 0 для привязанных к схеме ссылок, а также для межбазовых и межсерверных ссылок, которые явно указывают имя схемы. Например, ссылка на сущность в формате `EXEC MyDatabase.MySchema.MyProc` не зависит от вызывающего объекта. При этом ссылка в формате `EXEC MyDatabase..MyProc` зависит от вызывающего объекта.

`is_ambiguous` bit

Указывает, что ссылка является неоднозначной и на этапе выполнения может разрешиться к определяемой пользователем функции, определяемому пользователем типу или ссылке `XQuery` на столбец типа `xml`.

Например, предположим, что инструкция `SELECT Sales.GetOrder() FROM Sales.MySales` определена в хранимой процедуре. До выполнения хранимой процедуры неизвестно, является ли `Sales.GetOrder()` определяемой пользователем функцией в схеме `Sales` или столбцом `Sales` определяемого

пользователем типа с методом `GetOrder()`.

1 = Ссылка неоднозначна.

0 = Ссылка однозначна, или сущность можно успешно привязать при вызове представления.

Всегда принимает значение 0 для привязанных к схеме ссылок.

### Замечания

В следующей таблице перечислены типы сущностей, для которых созданы и обновляются данные о зависимостях. Данные о зависимостях не создаются и не обновляются для правил, значений по умолчанию, временных таблиц, временных хранимых процедур и системных объектов.

| Тип сущности                                    | Ссылающаяся сущность | Упоминаемая сущность |
|-------------------------------------------------|----------------------|----------------------|
| Таблица                                         | Да*                  | Да                   |
| Просмотр                                        | Да                   | Да                   |
| Фильтруемый индекс                              | Да**                 | Нет                  |
| Статистика фильтрации                           | Да**                 | Нет                  |
| Хранимая процедура Transact-SQL***              | Да                   | Да                   |
| CLR, хранимая процедура                         | Нет                  | Да                   |
| Определяемая пользователем функция Transact-SQL | Да                   | Да                   |
| Определяемая пользователем функция среды CLR    | Нет                  | Да                   |
| Триггер CLR (DML и DDL)                         | Нет                  | Нет                  |
| Триггер DML Transact-SQL                        | Да                   | Нет                  |
| Триггер DDL Transact-SQL уровня базы данных     | Да                   | Нет                  |
| Триггер DDL Transact-SQL уровня сервера         | Да                   | Нет                  |
| Расширенные хранимые процедуры                  | Нет                  | Да                   |



|                                                             |     |    |
|-------------------------------------------------------------|-----|----|
| Очередь                                                     | Нет | Да |
| Синоним                                                     | Нет | Да |
| Тип (псевдоним и определяемый пользователем тип данных CLR) | Нет | Да |
| Коллекция схем XML                                          | Нет | Да |
| Функция секционирования                                     | Нет | Да |

\* Таблица отслеживается в качестве ссылающейся сущности, только если она ссылается на модуль Transact-SQL, определяемый пользователем тип или коллекцию XML-схем в определении вычисляемого столбца, ограничении CHECK или ограничении DEFAULT.

\*\*Каждый столбец, используемый в предикате фильтра, отслеживается как ссылающаяся сущность.

\*\*\* Пронумерованные хранимые процедуры с целочисленным значением больше 1 не отслеживаются в качестве ссылающихся или упоминаемых сущностей.

Дополнительные сведения см. в разделе [Основные сведения о зависимостях SQL](#).

## Разрешения

Необходимо разрешение VIEW DEFINITION в базе данных и разрешение SELECT на представление sys.sql\_expression\_dependencies в базе данных. По умолчанию разрешение SELECT предоставляется только членам предопределенной роли базы данных db\_owner. Если разрешения SELECT и VIEW DEFINITION предоставлены другому пользователю, он может просматривать все зависимости в базе данных.

## Примеры

### А. Возвращение сущностей, на которые ссылаются другие сущности

В следующем примере возвращаются таблицы и столбцы, на которые ссылается представление Production.vProductAndDescription. Это представление зависит от сущностей (таблиц и столбцов), возвращаемых в столбцах referenced\_entity\_name и referenced\_column\_name.

```
USE AdventureWorks2008R2;
GO
SELECT OBJECT_NAME(referencing_id) AS referencing_entity_name,
       o.type_desc AS referencing_description,
       COALESCE(COL_NAME(referencing_id, referencing_minor_id), '(n/a)') AS
referencing_minor_id,
       referencing_class_desc, referenced_class_desc,
       referenced_server_name, referenced_database_name, referenced_schema_name,
       referenced_entity_name,
       COALESCE(COL_NAME(referenced_id, referenced_minor_id), '(n/a)') AS
referenced_column_name,
       is_caller_dependent, is_ambiguous
FROM sys.sql_expression_dependencies AS sed
```

```
INNER JOIN sys.objects AS o ON sed.referenceing_id = o.object_id
WHERE referenceing_id = OBJECT_ID(N'Production.vProductAndDescription');
GO
```

### Б. Возвращение сущностей, ссылающихся на другую сущность

В следующем примере возвращаются сущности, ссылающиеся на таблицу

Production.Product. Сущности, возвращенные в столбце referenceing\_entity\_name, зависят от таблицы Product.

```
USE AdventureWorks2008R2;
GO
SELECT OBJECT_SCHEMA_NAME ( referenceing_id ) AS referenceing_schema_name,
       OBJECT_NAME(referenceing_id) AS referenceing_entity_name,
       o.type_desc AS referenceing_description,
       COALESCE(COL_NAME(referenceing_id, referenceing_minor_id), '(n/a)') AS
referenceing_minor_id,
       referenceing_class_desc, referenced_class_desc,
       referenced_server_name, referenced_database_name, referenced_schema_name,
       referenced_entity_name,
       COALESCE(COL_NAME(referenced_id, referenced_minor_id), '(n/a)') AS
referenced_column_name,
       is_caller_dependent, is_ambiguous
FROM sys.sql_expression_dependencies AS sed
INNER JOIN sys.objects AS o ON sed.referenceing_id = o.object_id
WHERE referenced_id = OBJECT_ID(N'Production.Product');
GO
```

### В. Возвращение межбазовых зависимостей

В следующем примере возвращаются все межбазовые зависимости. Вначале в примере создается база данных db1 и две хранимые процедуры, которые ссылаются на таблицы в базах данных db2 и db3. Затем запрашивается таблица sys.sql\_expression\_dependencies, чтобы сообщить о наличии межбазовых зависимостей между процедурами и таблицами. Обратите внимание, что в столбце referenced\_schema\_name для упоминаемой сущности t3 возвращается значение NULL, потому что для этой сущности в определении процедуры не указано имя схемы.

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

## Функция динамического управления sys.dm\_sql\_referenced\_entities (Transact-SQL)

Возвращает по одной строке для каждой определяемой пользователем сущности, на имя которой ссылается определение рассматриваемой сущности. Зависимость между двумя сущностями создается в тех случаях, когда имя некоторой пользовательской сущности, называемой упоминаемой, встречается в составе постоянного выражения SQL другой пользовательской сущности, называемой ссылающейся. Например, если в качестве ссылающейся сущности выступает хранимая процедура, то данная функция выводит список всех определяемых пользователем сущностей, упоминаемых в данной хранимой процедуре. К упоминаемым определяемым пользователем сущностям относятся: таблицы, представления, определяемые пользователем типы (UDT), а также другие хранимые процедуры.

Данная функция динамического управления может быть использована для отображения списка сущностей, упоминаемых заданной ссылающейся сущностью и имеющих следующие типы:

- сущности, привязанные к схеме;
- несвязанные сущности;
- межбазовые и межсерверные сущности;
- зависимости уровня столбца, как связанные, так и не связанные со схемами;
- определяемые пользователем типы (псевдонимы и типы CLR);
- коллекции XML-схем;
- функции секционирования.

### Синтаксис

```
sys.dm_sql_referenced_entities (
    ' [ schema_name. ] referencing_entity_name ' , ' <referencing_class> ' )

<referencing_class> ::=
{
    OBJECT
  | DATABASE_DDL_TRIGGER
  | SERVER_DDL_TRIGGER
}
```

### Аргументы

[ schema\_name. ] referencing\_entity\_name

Имя ссылающейся сущности. Если ссылающаяся сущность принадлежит к классу объектов (OBJECT), то необходимо указывать аргумент schema\_name.

Аргумент schema\_name.referencing\_entity\_name имеет тип nvarchar(517).

<referencing\_class> ::= { OBJECT | DATABASE\_DDL\_TRIGGER | SERVER\_DDL\_TRIGGER }

Класс заданной ссылающейся сущности. В одной инструкции может быть указан только один класс.

Аргумент <referencing\_class> имеет тип nvarchar(60).

### Возвращенная таблица

| Имя столбца              | Тип данных | Описание                                                                                                                                                                                                                                                     |
|--------------------------|------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| referencing_minor_id     | int        | Идентификатор столбца, если ссылающаяся сущность является столбцом; в противном случае — 0. Не допускает значение NULL.                                                                                                                                      |
|                          |            | Имя сервера упоминаемой сущности.                                                                                                                                                                                                                            |
|                          |            | Этот столбец заполняется для межсерверных зависимостей, которые создаются путем задания допустимого четырехкомпонентного имени. Сведения о многокомпонентных именах см. в разделе <a href="#">Синтаксические обозначения в Transact-SQL (Transact-SQL)</a> . |
| referenced_server_name   | sysname    | Имеет значение NULL для не привязанных к схеме зависимостей, в которых сущность упоминается без указания четырехкомпонентного имени.                                                                                                                         |
|                          |            | Имеет значение NULL для привязанных к схеме сущностей, поскольку они должны находиться в одной базе данных, и поэтому их можно определить только с использованием только двухкомпонентного имени (схема.объект).                                             |
|                          |            | Имя базы данных упоминаемой сущности.                                                                                                                                                                                                                        |
|                          |            | Этот столбец заполняется для межбазовых и межсерверных ссылок, которые задаются путем указания допустимого трехкомпонентного или четырехкомпонентного имени.                                                                                                 |
| referenced_database_name | sysname    | Имеет значение NULL для не привязанных к схеме ссылок, задаваемых с помощью однокомпонентного или двухкомпонентного имени.                                                                                                                                   |
|                          |            | Имеет значение NULL для привязанных к схеме сущностей, поскольку они должны находиться в одной базе данных, и поэтому их можно определить только с использованием только двухкомпонентного имени (схема.объект).                                             |
| referenced_schema_name   | sysname    | Схема, которой принадлежит упоминаемая сущность.                                                                                                                                                                                                             |

|                        |         |                                                                                                                                                                                                                                                                                                                                                                                                                                          |
|------------------------|---------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|                        |         | Имеет значение NULL для не привязанных к схеме ссылок, в которых сущность упоминается без указания имени схемы.                                                                                                                                                                                                                                                                                                                          |
|                        |         | Никогда не принимает значение NULL для привязанных к схеме ссылок.                                                                                                                                                                                                                                                                                                                                                                       |
| referenced_entity_name | sysname | Имя упоминаемой сущности. Не допускает значение NULL.                                                                                                                                                                                                                                                                                                                                                                                    |
| referenced_minor_name  | sysname | Имя столбца, если упоминаемая сущность является столбцом. В противном случае — значение NULL. Например, параметр referenced_minor_name будет иметь значение NULL в строке, которая содержит саму упоминаемую сущность.                                                                                                                                                                                                                   |
|                        |         | Упоминаемая сущность представляет собой столбец, если в ссылающейся сущности столбец определяется по имени или если в инструкции SELECT * используется родительская сущность.                                                                                                                                                                                                                                                            |
|                        |         | Идентификатор упоминаемой сущности. Если значение параметра referenced_minor_id отлично от 0, то величина referenced_id представляет сущность, определяющую столбец.                                                                                                                                                                                                                                                                     |
|                        |         | Для межсерверных ссылок всегда принимает значение NULL.                                                                                                                                                                                                                                                                                                                                                                                  |
|                        |         | Принимает значение NULL для межбазовых ссылок в тех случаях, когда не удастся определить идентификатор базы данных вне сети или неограниченности сущности.                                                                                                                                                                                                                                                                               |
| referenced_id          | int     | Имеет значение NULL для ссылок в пределах базы данных, когда не удастся определить идентификатор. Для ссылок, не привязанных к схеме, идентификатор не удастся разрешить в следующих случаях. <ul style="list-style-type: none"> <li>• Упоминаемая сущность не существует в базе данных.</li> <li>• Разрешение имени зависит от вызывающего объекта. В этом случае параметр is_caller_dependent устанавливается в значение 1.</li> </ul> |
|                        |         | Никогда не принимает значение NULL для привязанных к схеме ссылок.                                                                                                                                                                                                                                                                                                                                                                       |
| referenced_minor_id    | int     | Идентификатор столбца, если упоминаемая сущность является столбцом. В противном случае — 0. Например, параметр referenced_minor_id будет иметь значение 0 в строке, которая содержит саму упоминаемую сущность.                                                                                                                                                                                                                          |
|                        |         | При наличии не привязанных к схеме ссылок список зависимостей уровня столбцов может быть выведен только в случае ограниченности всех упоминаемых сущностей. Если                                                                                                                                                                                                                                                                         |

какая-либо из упоминаемых сущностей не может быть привязана, в списке не отображается ни одной зависимости уровня столбца, а параметру `referenced_minor_id` присваивается значение 0. См. пример Г.

Класс упоминаемой сущности.

1 = Объект или столбец.

`referenced_class` tinyint

6 = Тип.

10 = Коллекция схем XML.

21 = Функция секционирования.

Описание класса упоминаемой сущности.

OBJECT\_OR\_COLUMN

`referenced_class_desc` nvarchar(60)

TYPE

XML\_SCHEMA\_COLLECTION

PARTITION\_FUNCTION

Отображает привязку к схеме для упоминаемой сущности, полученную во время выполнения (так как идентификатор сущности зависит от схемы вызывающего объекта). Данная ситуация возможна только для хранимой процедуры, расширенной хранимой процедуры или определяемой пользователем функции, выполняемых в инструкции EXECUTE.

`is_caller_dependent` bit

1 = упоминаемая сущность зависит от вызывающего объекта и определяется во время выполнения. В этом случае параметр `referenced_id` принимает значение NULL.

0 = Идентификатор упоминаемой сущности не зависит от вызывающего объекта. Всегда имеет значение 0 для привязанных к схеме ссылок, а также для межбазовых и межсерверных ссылок, которые явно указывают имя схемы. Например, ссылка на сущность в формате EXEC MyDatabase.MySchema.MyProc не зависит от вызывающего объекта. При этом ссылка в формате EXEC MyDatabase..MyProc зависит от вызывающего объекта.

`is_ambiguous` bit

Указывает, что ссылка является неоднозначной во время выполнения может разрешиться к определяемой пользователем функции, определяемому пользователем типу или ссылке XQuery на столбец типа xml. Например, предположим, что инструкция SELECT Sales.GetOrder() FROM Sales.MySales определяется в хранимой процедуре. До выполнения хранимой процедуры неизвестно, является ли Sales.GetOrder()

определяемой пользователем функцией в схеме Sales или столбцом Sales определяемого пользователем типа с методом GetOrder().

1 = ссылка на определяемую пользователем функцию или на метод определяемого пользователем типа (UDT) столбца неоднозначна.

0 = ссылка определена однозначно, либо сущность при вызове функции может быть привязана.

Для привязанных к схеме ссылок всегда принимает значение 0.

### Исключения

Возвращает пустой результирующий набор, если выполняется любое из следующих условий.

- Указан системный объект.
- Указанная сущность не существует в текущей базе данных.
- Указанная сущность не ссылается ни на какие сущности.
- Передан недопустимый параметр.

Выдает ошибку, если заданная ссылающаяся сущность является пронумерованной хранимой процедурой.

Возвращает ошибку 2020, когда не удастся разрешить зависимости столбца. Эта ошибка не препятствует возврату запросом зависимостей на уровне объектов. Дополнительные сведения см. в разделе [Диагностика зависимостей SQL](#).

### Замечания

Данная функция может выполняться в контексте любой базы данных и осуществляет отображение списка сущностей, ссылающихся на триггер DDL уровня сервера.

В следующей таблице перечислены типы сущностей, для которых созданы и обновляются данные о зависимостях. Данные о зависимостях не создаются и не обновляются для правил, значений по умолчанию, временных таблиц, временных хранимых процедур и системных объектов.

| Тип сущности                      | Ссылающаяся сущность | Упомянутая сущность |
|-----------------------------------|----------------------|---------------------|
| Таблица                           | Да*                  | Да                  |
| Представление                     | Да                   | Да                  |
| Хранимая процедура Transact-SQL** | Да                   | Да                  |

|                                                             |     |     |
|-------------------------------------------------------------|-----|-----|
| Хранимая процедура среды CLR                                | Нет | Да  |
| Определяемая пользователем функция Transact-SQL             | Да  | Да  |
| Определяемая пользователем функция среды CLR                | Нет | Да  |
| Триггер CLR (DML и DDL)                                     | Нет | Нет |
| Триггер DML Transact-SQL                                    | Да  | Нет |
| Триггер DDL Transact-SQL уровня базы данных                 | Да  | Нет |
| Триггер DDL Transact-SQL уровня сервера                     | Да  | Нет |
| Расширенные хранимые процедуры                              | Нет | Да  |
| Очередь                                                     | Нет | Да  |
| Синоним                                                     | Нет | Да  |
| Тип (псевдоним и определяемый пользователем тип данных CLR) | Нет | Да  |
| Коллекция схем XML                                          | Нет | Да  |
| Функция секционирования                                     | Нет | Да  |

\* Таблица отслеживается в качестве ссылающейся сущности, только если она ссылается на модуль Transact-SQL, определяемый пользователем тип или коллекцию схем XML в определении вычисляемого столбца, ограничении CHECK или ограничении DEFAULT.

\*\* Пронумерованные хранимые процедуры с целочисленным значением больше 1 не отслеживаются в качестве ссылающихся или упоминаемых сущностей.

Дополнительные сведения см. в разделе [Основные сведения о зависимостях SQL](#).

### **Разрешения**

Требуется разрешения SELECT для функции sys.dm\_sql\_referenced\_entities и разрешения VIEW DEFINITION для ссылающейся сущности. Разрешение SELECT по умолчанию предоставляется роли public. Требуется разрешения VIEW DEFINITION для базы данных, либо, если ссылающаяся сущность является триггером DDL уровня базы данных, разрешения ALTER DATABASE DDL TRIGGER. Если указанный модуль является



триггером DDL уровня сервера, то требуется разрешение VIEW ANY DEFINITION на уровне сервера.

## Примеры

### А. Отображение списка сущностей, упоминаемых триггером DDL уровня базы данных

В ходе выполнения следующего примера производится отображение списка сущностей (таблиц и столбцов), упоминаемых триггером DDL уровня базы данных ddlDatabaseTriggerLog.

```
USE AdventureWorks2008R2;
GO
SELECT referenced_schema_name, referenced_entity_name, referenced_minor_name,
       referenced_minor_id, referenced_class_desc
FROM sys.dm_sql_referenced_entities ('ddlDatabaseTriggerLog',
'DATABASE_DDL_TRIGGER');
```

### Б. Отображение списка сущностей, упоминаемых объектом

В ходе выполнения следующего примера производится отображение списка сущностей, упоминаемых в определяемой пользователем функции dbo.ufnGetContactInformation.

```
USE AdventureWorks2008R2;
GO
SELECT referenced_schema_name, referenced_entity_name, referenced_minor_name,
       referenced_minor_id, referenced_class_desc, is_caller_dependent,
is_ambiguous
FROM sys.dm_sql_referenced_entities ('dbo.ufnGetContactInformation', 'OBJECT');
```

### В. Отображение списка зависимостей столбцов

В ходе выполнения представленного ниже примера осуществляется создание таблицы Table1, в которой содержится вычисляемый столбец c, определяемый как сумма столбцов a и b. Затем производится вызов представления sys.dm\_sql\_referenced\_entities. Представление содержит две строки, по одной для каждого столбца, определенного в вычисляемом столбце.

```
USE AdventureWorks2008R2;
GO
CREATE TABLE dbo.Table1 (a int, b int, c AS a + b);
GO
SELECT referenced_schema_name AS schema_name,
       referenced_entity_name AS table_name,
       referenced_minor_name AS referenced_column,
       COALESCE(COL_NAME(OBJECT_ID(N'dbo.Table1'),referencing_minor_id), 'N/A') AS
referencing_column_name
FROM sys.dm_sql_referenced_entities ('dbo.Table1', 'OBJECT');
```

-- Remove the table.  
DROP TABLE dbo.Table1;

```
GO
Ниже приводится результирующий набор.

schema_name table_name referenced_column referencing_column
-----
dbo Table1 a c
```

```
dbo Table1 b c
```

### Г. Отображение зависимостей столбцов, не привязанных к схеме

В ходе выполнения представленного ниже примера производится очистка таблицы Table1, а также создание таблицы Table2 и хранимой процедуры Proc1. Процедура ссылается на таблицу Table2 и на несуществующую таблицу Table1. Представление sys.dm\_sql\_referenced\_entities запускается с помощью хранимой процедуры, которая указана в качестве ссылающейся сущности. В результирующем наборе показана одна строка для таблиц Table1 и Table2. Поскольку таблица Table1 не существует, то не удастся разрешить зависимости столбца и возвращается ошибка 2020.

```
USE AdventureWorks2008R2;
GO
IF OBJECT_ID ( 'dbo.Table1', 'U' ) IS NOT NULL
    DROP TABLE dbo.Table1;
GO
CREATE TABLE dbo.Table2 (c1 int, c2 int);
GO
CREATE PROCEDURE dbo.Proc1 AS
    SELECT a, b, c FROM Table1;
    SELECT c1, c2 FROM Table2;
GO
SELECT referenced_id, referenced_entity_name AS table_name,
    referenced_minor_name AS referenced_column_name
FROM sys.dm_sql_referenced_entities ('dbo.Proc1', 'OBJECT');
GO
```

### Д. Отображение содержимого динамических зависимостей

Следующий пример является продолжением примера Г и предназначен для отображения динамических зависимостей. В примере сначала производится повторное создание таблицы Table1, которая была удалена в примере Г. Затем снова запускается инструкция sys.dm\_sql\_referenced\_entities, в которой в качестве ссылающейся сущности указана хранимая процедура. Результирующий набор содержит обе таблицы и их соответствующие столбцы, определенные в хранимой процедуре.

```
USE AdventureWorks2008R2;
GO
CREATE TABLE Table1 (a int, b int, c AS a + b);
GO
SELECT referenced_id, referenced_entity_name AS table_name,
    referenced_minor_name as column_name
FROM sys.dm_sql_referenced_entities ('dbo.Proc1', 'OBJECT');
GO
DROP TABLE Table1, Table2;
DROP PROC Proc1;
GO
```

Ниже приводится результирующий набор.

```
referenced_id, table_name, column_name
-----
2139154566      Table1      NULL
2139154566 Table1 a
2139154566 Table1 b
2139154566 Table1 c
2707154552 Table2 NULL
2707154552 Table2 c1
```

2707154552 Table2 c2

## sys.dm\_sql\_referencing\_entities (Transact-SQL)

Возвращает одну строку для каждой сущности текущей базы данных, которая ссылается на имя какой-либо другой определенной пользователем сущности. Зависимость между двумя сущностями создается, когда имя некоторой сущности, называемой упоминаемой сущностью, встречается в составе постоянного выражения языка SQL другой сущности, называемой ссылающейся сущностью. Например, если в качестве упоминаемой сущности, выступает определяемый пользователем тип (UDT), то данная функция возвращает список всех определяемых пользователем сущностей, ссылающихся в своих определениях на имя указанного определяемого пользователем типа. Функция не возвращает список сущностей других баз данных, ссылающихся на указанную сущность. Данная функция предназначена для выполнения в контексте базы данных master и возвращает триггер DDL уровня сервера как ссылающаяся сущность.

Данная динамическая административная функция может быть использована для отображения списка сущностей текущей базы данных, ссылающихся на заданную сущность и имеющих следующие типы:

- сущности, как связанные со схемой, так и не связанные;
- триггеры DDL уровня базы данных;
- триггеры DDL уровня сервера.

### Синтаксис

```
sys.dm_sql_referencing_entities (
    ' schema_name.referenced_entity_name ' , ' <referenced_class> ' )
```

```
<referenced_class> ::=
{
    OBJECT
  | TYPE
  | XML_SCHEMA_COLLECTION
  | PARTITION_FUNCTION
}
```

### Аргументы

schema\_name.referenced\_entity\_name

Имя упоминаемой сущности.

Аргумент schema\_name необходим, кроме тех случаев, когда упоминаемый класс относится к классу PARTITION\_FUNCTION.

Аргумент schema\_name.referenced\_entity\_name имеет тип nvarchar(517).

```
<упоминаемый_класс> ::= { OBJECT
  | TYPE  | XML_SCHEMA_COLLECTION | PARTITION_FUNCTION }
```

Класс упоминаемой сущности. В одной инструкции может быть указан только один класс.

Параметр <referenced\_class> имеет тип nvarchar(60).

### **Возвращенная таблица**

| Имя столбца             | Тип данных   | Описание                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
|-------------------------|--------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| referencing_schema_name | sysname      | <p>Схема, которой принадлежит ссылающаяся сущность. Допускает значения NULL.</p> <p>Значение NULL для триггеров DDL уровня базы данных или сервера.</p>                                                                                                                                                                                                                                                                                                                                                                                    |
| referencing_entity_name | sysname      | Имя ссылающейся сущности. Не допускает значение NULL.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| referencing_id          | int          | Идентификатор ссылающейся сущности. Не допускает значения NULL.                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| referencing_class       | tinyint      | <p>Класс ссылающейся сущности. Не допускает значения NULL.</p> <p>1 = Объект</p> <p>12 = Триггер DDL уровня базы данных</p> <p>13 = Триггер DDL уровня сервера</p> <p>Описание класса ссылающейся сущности.</p>                                                                                                                                                                                                                                                                                                                            |
| referencing_class_desc  | nvarchar(60) | <p>OBJECT</p> <p>DATABASE_DDL_TRIGGER</p> <p>SERVER_DDL_TRIGGER</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| is_caller_dependent     | bit          | <p>Указывает разрешение идентификатора упоминаемой сущности, полученного во время выполнения (так как он зависит от схемы вызывающего объекта).</p> <p>Значение 1 означает, что ссылающаяся сущность может ссылаться на данный объект. При этом разрешение упоминаемой сущности зависит от вызывающего объекта и не может быть определено. Данная ситуация возможна только при вызове в инструкции EXECUTE не связанной со схемой ссылки на хранимую процедуру, расширенную хранимую процедуру или определяемую пользователем функцию.</p> |

Значение 0 означает, что упоминаемая сущность не зависит от вызывающего объекта.

### Исключения

Возвращает пустой результирующий набор, если выполняется любое из следующих условий.

- Указан системный объект.
- Указанная сущность не существует в текущей базе данных.
- Указанная сущность не ссылается ни на какие сущности.
- Передан недопустимый параметр.

Возвращает ошибку, если заданная упоминаемая сущность является пронумерованной хранимой процедурой.

### Замечания

В следующей таблице перечислены типы сущностей, для которых созданы и обновляются данные о зависимостях. Данные о зависимостях не создаются и не обновляются для правил, значений по умолчанию, временных таблиц, временных хранимых процедур и системных объектов.

| Тип сущности                                    | Ссылающаяся сущность | Упоминаемая сущность |
|-------------------------------------------------|----------------------|----------------------|
| Таблица                                         | Да*                  | Да                   |
| Представление                                   | Да                   | Да                   |
| Хранимая процедура Transact-SQL**               | Да                   | Да                   |
| Хранимая процедура CLR                          | Нет                  | Да                   |
| Определяемая пользователем функция Transact-SQL | Да                   | Да                   |
| Определяемая пользователем функция CLR          | Нет                  | Да                   |
| Триггер CLR (DML и DDL)                         | Нет                  | Нет                  |
| Триггер DML Transact-SQL                        | Да                   | Нет                  |
| Триггер DDL Transact-SQL уровня базы данных     | Да                   | Нет                  |

|                                                             |     |     |
|-------------------------------------------------------------|-----|-----|
| Триггер DDL Transact-SQL уровня сервера                     | Да  | Нет |
| Расширенные хранимые процедуры                              | Нет | Да  |
| Очередь                                                     | Нет | Да  |
| Синоним                                                     | Нет | Да  |
| Тип (псевдоним и определяемый пользователем тип данных CLR) | Нет | Да  |
| Коллекция схем XML                                          | Нет | Да  |
| Функция секционирования                                     | Нет | Да  |

\* Таблица отслеживается в качестве ссылающейся сущности, только если она ссылается на модуль Transact-SQL, определяемый пользователем тип или коллекцию схем XML в определении вычисляемого столбца, ограничении CHECK или ограничении DEFAULT.

\*\* Пронумерованные хранимые процедуры с целочисленным значением больше 1 не отслеживаются в качестве ссылающихся или упоминаемых сущностей.

Дополнительные сведения см. в разделе [Основные сведения о зависимостях SQL](#).

## Разрешения

Необходимо разрешение CONTROL на упоминаемую сущность и разрешение SELECT на представление sys.dm\_sql\_referencing\_entities. Если упоминаемая сущность является функцией секционирования, необходимо разрешение CONTROL на базу данных. Разрешение SELECT по умолчанию предоставляется роли public.

## Примеры

### А. Получение списка сущностей, ссылающихся на заданную сущность

В следующем примере возвращается список сущностей текущей базы данных, которые ссылаются на указанную таблицу.

```
USE AdventureWorks2008R2;
GO
SELECT referencing_schema_name, referencing_entity_name, referencing_id,
referencing_class_desc, is_caller_dependent
FROM sys.dm_sql_referencing_entities ('Production.Product', 'OBJECT');
```

### Б. Получение списка сущностей, ссылающихся на заданный тип

В следующем примере возвращается список сущностей, ссылающихся на псевдоним типа dbo.Flag. Результирующий набор показывает, что этот тип используется двумя хранимыми процедурами. Тип dbo.Flag также используется в определениях некоторых столбцов таблицы HumanResources.Employee. Однако для таблицы HumanResources.Employee не выдается никаких строк, так как указанный тип не

содержится в определении вычисляемого столбца, ограничения CHECK или DEFAULT для данной таблицы.

```
USE AdventureWorks2008R2;
GO
SELECT referencing_schema_name, referencing_entity_name, referencing_id,
referencing_class_desc, is_caller_dependent
FROM sys.dm_sql_referencing_entities ('dbo.Flag', 'TYPE');
GO
```

Ниже приводится результирующий набор.

```
referencing_schema_name referencing_entity_name referencing_id
referencing_class_desc is_caller_dependent
-----
HumanResources uspUpdateEmployeeHireInfo 1803153469 OBJECT_OR_COLUMN 0
HumanResources uspUpdateEmployeeLogin 1819153526 OBJECT_OR_COLUMN 0
(Обработано строк: 2)
```

## Удаление таблицы

Иногда бывает нужно удалить таблицу; например, может понадобиться реализовать новую структуру или освободить место в базе данных. При удалении таблицы ее структурное определение, данные, полнотекстовые индексы, ограничения и индексы удаляются из базы данных навсегда, а место, используемое ранее для хранения таблицы и ее индексов, становится доступным для других таблиц. Временную таблицу можно удалить явно, не дожидаясь ее автоматического удаления.

Если нужно удалить таблицы, связанные посредством ограничений FOREIGN KEY и UNIQUE либо PRIMARY KEY, сначала необходимо удалить таблицы с ограничениями FOREIGN KEY. Если нужно удалить таблицу, на которую ссылается ограничение FOREIGN KEY, но удалить всю таблицу с внешним ключом невозможно, необходимо удалить ограничение FOREIGN KEY.

Если нужно удалить все данные из таблицы, но не саму таблицу, можно выполнить ее усечение. Дополнительные сведения см. в разделе [Удаление всех строк с помощью инструкции TRUNCATE TABLE](#).

Можно удалять или выполнять усечение больших таблиц и индексов, которые используют более чем 128 экстендов, без удержания одновременной блокировки всех кластеров страниц, необходимых для удаления. Компонент Database Engine удаляет большие объекты за две различные фазы: логический и физический. В логической фазе существующие единицы распределения, используемые таблицей, помечаются для освобождения и остаются заблокированными до фиксации транзакции. В физической фазе единицы распределения, помеченные для освобождения, физически удаляются пакетами. Дополнительные сведения см. в разделе [Удаление и повторная сборка больших объектов](#).

## DROP TABLE (Transact-SQL)

Удаляет одно или больше определений таблиц и все данные, индексы, триггеры,

ограничения и разрешения для этих таблиц. Любое представление или хранимая процедура, ссылающаяся на удаленную таблицу, должна быть явно удалена с помощью инструкции [DROP VIEW](#) или [DROP PROCEDURE](#). Получить отчет о зависимостях в таблице можно с помощью [sys.dm\\_sql\\_referencing\\_entities](#).

### *Синтаксис*

```
DROP TABLE [ database_name . [ schema_name ] . | schema_name . ]  
            table_name [ ,...n ] [ ; ]
```

### *Аргументы*

database\_name

Имя базы данных, в которой создана таблица.

schema\_name

Имя схемы, которой принадлежит таблица.

table\_name

Имя таблицы, предназначенной для удаления.

### *Замечания*

Инструкцию DROP TABLE нельзя использовать для удаления таблицы, на которую ссылается ограничение FOREIGN KEY. Сначала следует удалить ссылающееся ограничение FOREIGN KEY или ссылающуюся таблицу. Если и ссылающаяся таблица, и таблица, содержащая первичный ключ, удаляются с помощью одной инструкции DROP TABLE, ссылающаяся таблица должна быть первой в списке.

Несколько таблиц можно удалить из любой базы данных. Если удаляемая таблица ссылается на первичный ключ другой таблицы, которая также удаляется, ссылающаяся таблица с внешним ключом должна стоять в списке перед таблицей, содержащей указанный первичный ключ.

При удалении таблицы относящиеся к ней правила и значения по умолчанию теряют привязку, а любые связанные с таблицей ограничения или триггеры автоматически удаляются. Если таблица будет создана заново, нужно будет заново привязать все правила и значения по умолчанию, заново создать триггеры и добавить необходимые ограничения.

При удалении всех строк в таблице с помощью инструкции DELETE tablename или TRUNCATE TABLE таблица продолжает существовать, пока она не будет удалена.

Большие таблицы и индексы из более чем 128 экстенгов удаляются в два этапа: логический и физический. На логическом этапе существующие единицы распределения, используемые в таблице, отмечаются для освобождения и остаются заблокированными до фиксации транзакции. На физическом этапе страницы IAM, отмеченные для



освобождения, физически удаляются пакетами. Дополнительные сведения см. в разделе [Удаление и повторная сборка больших объектов](#).

При удалении таблицы, которая содержит столбец VARBINARY(MAX) с атрибутом FILESTREAM, не будут удалены никакие данные, которые хранятся в файловой системе.

### **Важно!**

Инструкции DROP TABLE и CREATE TABLE нельзя выполнять для одной таблицы в одном пакете. В противном случае может произойти непредвиденная ошибка.

### **Разрешения**

Требует разрешения на работу с ALTER в схеме, к которой принадлежит эта таблица, разрешения на работу с CONTROL для этой таблицы или членства в предопределенной роли базы данных db\_ddladmin.

### **Примеры**

#### **А. Удаление таблицы из текущей базы данных**

Следующий пример удаляет таблицу ProductVendor1, ее данные и индексы из текущей базы данных.

```
DROP TABLE ProductVendor1 ;
```

#### **Б. Удаление таблицы из другой базы данных**

Следующий пример удаляет таблицу SalesPerson2 из базы данных База данных AdventureWorks2008R2. Пример может быть выполнен из любой базы данных на экземпляре сервера.

```
DROP TABLE AdventureWorks2008R2.dbo.SalesPerson2 ;
```

#### **В. Удаление временной таблицы**

Следующий пример создает временную таблицу, проверяет ее наличие, удаляет ее и снова проверяет ее наличие.

```
USE AdventureWorks2008R2;
GO
CREATE TABLE #temptable (col1 int);
GO
INSERT INTO #temptable
VALUES (10);
GO
SELECT * FROM #temptable;
GO
IF OBJECT_ID(N'tempdb..#temptable', N'U') IS NOT NULL
DROP TABLE #temptable;
GO
--Test the drop.
SELECT * FROM #temptable;
```

## TRUNCATE TABLE (Transact-SQL)

Удаляет все строки в таблице, не записывая в журнал удаление отдельных строк. Инструкция TRUNCATE TABLE похожа на инструкцию DELETE без предложения WHERE, однако TRUNCATE TABLE выполняется быстрее и требует меньших ресурсов системы и журналов транзакций.

### Синтаксис

```
TRUNCATE TABLE
    [ { database_name . [ schema_name ] . | schema_name . } ]
    table_name
[ ; ]
```

### Аргументы

database\_name

Имя базы данных.

schema\_name

Имя схемы, которой принадлежит таблица.

table\_name

Имя таблицы, которая должна быть усечена, или таблицы, из которой удаляются все строки.

### Замечания

По сравнению с инструкцией DELETE, инструкция TRUNCATE TABLE обладает следующими преимуществами:

- Используется меньший объем журнала транзакций.

Инструкция DELETE производит удаление по одной строке и заносит в журнал транзакций запись для каждой удаляемой строки. Инструкция TRUNCATE TABLE удаляет данные, освобождая страницы данных, используемые для хранения данных таблиц, и в журнал транзакций записывает только данные об освобождении страниц.

- Обычно используется меньшее количество блокировок.

Если инструкция DELETE выполняется с блокировкой строк, для удаления блокируется каждая строка таблицы. Инструкция TRUNCATE TABLE всегда блокирует таблицу и страницу, но не каждую строку.

- В таблице остается нулевое количество страниц, без исключений.

После выполнения инструкции DELETE в таблице могут все еще оставаться

пустые страницы. Например, чтобы освободить пустые страницы в куче, необходима, как минимум, монопольная блокировка таблицы (LCK\_M\_X). Если операция удаления не использует блокировку таблицы, таблица (куча) будет содержать множество пустых страниц. В индексах после операции удаления могут оказаться пустые страницы, хотя эти страницы будут быстро освобождены процессом фоновой очистки.

Инструкция TRUNCATE TABLE удаляет все строки таблицы, но структура таблицы и ее столбцы, ограничения, индексы и т. п. сохраняются. Чтобы удалить не только данные таблицы, но и ее определение, следует использовать инструкцию DROP TABLE.

Если таблица содержит столбец идентификаторов, счетчик этого столбца сбрасывается до начального значения, определенного для этого столбца. Если начальное значение не задано, используется значение по умолчанию, равное 1. Чтобы сохранить столбец идентификаторов, используйте инструкцию DELETE.

### **Ограничения**

Инструкцию TRUNCATE TABLE нельзя использовать для таблиц, для которых выполняются следующие условия:

- На таблицу ссылается ограничение FOREIGN KEY. (Таблицу, имеющую внешний ключ, ссылающийся сам на себя, можно усесть.)
- Таблица является частью индексируемого представления.
- Таблица опубликована с использованием репликации транзакций или репликации слиянием.

Для таблиц с какими-либо из этих характеристик следует использовать инструкцию DELETE.

Инструкция TRUNCATE TABLE не может активировать триггер, поскольку она не записывает в журнал удаление отдельных строк. Дополнительные сведения см. в разделе [CREATE TRIGGER \(Transact-SQL\)](#).

### **Усечение больших таблиц**

В Microsoft SQL Server существует возможность удалять или усекают таблицы, которые имеют больше 128 экстендов, не удерживая одновременные блокировки для всех экстендов, предназначенных для удаления. Дополнительные сведения см. в разделе [Удаление и повторная сборка больших объектов](#).

### **Разрешения**

Минимально необходимым разрешением является ALTER table\_name. Разрешения по умолчанию для инструкции TRUNCATE TABLE распространяются на владельца таблицы, членов предопределенной роли сервера sysadmin, а также предопределенных ролей базы данных db\_owner и db\_ddladmin. Эти разрешения не передаются. Тем не менее инструкцию TRUNCATE TABLE можно встроить в модуль, например в хранимую

процедуру, и предоставить соответствующие разрешения этому модулю с помощью предложения EXECUTE AS. Дополнительные сведения см. в разделе [Использование инструкции EXECUTE AS для создания пользовательских наборов разрешений](#).

### Примеры

В следующем примере удаляются все данные из таблицы JobCandidate. Для сравнения результатов до и после инструкции TRUNCATE TABLE включаются инструкции SELECT.

```
USE AdventureWorks2008R2;  
GO  
SELECT COUNT(*) AS BeforeTruncateCount  
FROM HumanResources.JobCandidate;  
GO  
TRUNCATE TABLE HumanResources.JobCandidate;  
GO  
SELECT COUNT(*) AS AfterTruncateCount  
FROM HumanResources.JobCandidate;  
GO
```

## Удаление и повторная сборка больших объектов

При удалении или перестроении больших индексов либо удалении или усечении больших таблиц компонент SQL Server 2005 Database Engine откладывает фактическое освобождение страниц и взаимосвязанных блокировок до момента фиксации транзакции. В многопользовательской среде поддерживаются как автоматически, так и явно фиксируемые транзакции, которые могут применяться к большим таблицам и индексам, состоящим более чем из 128 экстендов.

Компонент Database Engine избегает блокировки размещения, требуемой для удаления больших объектов, с помощью разделения процесса на две стадии: логическую и физическую.

На логическом этапе существующие единицы распределения, используемые в таблице, помечаются как освобождаемые и блокируются до момента фиксации транзакции. При удалении кластеризованного индекса строки данных копируются, а затем перемещаются в кучу или новое расположение, предназначенное для хранения перестроенных кластеризованных индексов. (В случае перестроения индексов строки данных дополнительно подвергаются сортировке.) При откате происходит откат только данной логической стадии.

Физическая стадия удаления начинается после фиксации транзакции. Единицы распределения, помеченные как освобождаемые, физически удаляются одним пакетом. Эти операции удаления проводятся в коротких транзакциях, выполняемых в фоновом режиме и не требующих большого количества памяти.

В силу того, что физическая стадия начинается после фиксации транзакции, дисковое пространство таблицы или индекса может оставаться недоступным в течение некоторого времени. Если указанное дисковое пространство требуется для расширения таблицы до завершения физической стадии удаления, компонент Database Engine пытается высвободить пространство, занятое единицами распределения, помеченными

как освобождаемые. Для определения пространства, занимаемого указанными единицами распределения в данный момент, используйте представление каталога [sys.allocation\\_units](#).

Отложенные операции удаления не сразу освобождают выделенное им пространство, что приводит к дополнительным временным затратам компонента Database Engine. Поэтому таблицы и индексы, занимающие менее 128 экстенгов, удаляются, усекаются и восстанавливаются такими же методами, что и в SQL Server 2000. То есть при их удалении логическая и физическая стадии заканчиваются до момента фиксации транзакции.

