



Заметки по книге Microsoft SQL Server 2012.
Создание запросов. Учебный курс Microsoft.
Ицик Бен-Ган, Деян Сарка, Рон Талмейдж

Составил Криков Антон

Москва — 2022 г.

Содержание

1 Основы построения запросов	9
1.1 Основы языка T-SQL	9
1.2 Понимание логической обработки запросов	12
2 Начало работы с инструкцией SELECT	18
2.1 Использование FROM и SELECT	18
2.2 Работа с типами данных и встроенными функциями	21
2.2.1 Выбор типов данных для ключей	22
2.2.2 Выражение CASE и связанные с ним функции	24
2.2.3 Функция COALESCE	24
3 Фильтрация и сортировка данных	32
3.1 Фильтрация данных с помощью предикатов	32
3.1.1 Предикат LIKE	33
3.1.2 Фильтрация данных даты и времени	34
3.2 Сортировка данных	38
3.3 Фильтрация данных с помощью TOP и OFFSET...FETCH . . .	42
3.3.1 Фильтрация данных с помощью предложения TOP . . .	42
3.3.2 Фильтрация данных с помощью OFFSET...FETCH . . .	43
4 Комбинирование наборов данных	53
4.1 Использование соединений	53
4.1.1 Перекрестные соединения CROSS JOIN	53
4.1.2 Внутренние соединения INNER JOIN	54
4.1.3 Внешние соединения OUTER JOIN	55
4.2 Использование подзапросов, табличных выражений и оператора APPLY	60
4.2.1 Независимые подзапросы	60
4.2.2 Коррелированные (связанные) подзапросы	60
4.3 Табличные выражения	61
4.3.1 Производные таблицы	62
4.3.2 Обобщенные табличные выражения	63
4.3.3 Представления и встроенные табличные функции	65

4.4	Оператор APPLY	67
4.4.1	CROSS APPLY	67
4.4.2	OUTER APPLY	68
4.5	Использование операторов работы с наборами	74
4.5.1	Операторы UNION и UNION ALL	75
4.5.2	Оператор INTERSECT	77
4.5.3	Оператор EXCEPT	77
5	Группирование и оконные функции	83
5.1	Написание запросов для группировки данных	83
5.2	Работа с несколькими наборами группирования	83
5.2.1	GROUPING SETS	84
5.2.2	CUBE	85
5.2.3	ROLLUP	85
5.2.4	GROUPING, GROUPING_ID	87
5.3	Сведение и отмена сведения данных	92
5.3.1	Сведение данных	92
5.3.2	Отмена сведения данных	93
5.4	Использование оконных функций	97
5.4.1	Статистические оконные функции	97
5.4.2	Ранжирующие оконные функции	98
5.4.3	Оконные функции смещения	100
6	Запросы с полнотекстовым поиском данных	107
6.1	Создание полнотекстовых каталогов и индексов	107
6.1.1	Компоненты полнотекстового поиска	107
6.2	Использование предикатов CONTAINS и FREETEXT	110
6.2.1	Предикат CONTAINS	110
6.2.2	Предикат FREETEXT	110
6.3	Использование табличных функций полнотекстового и семантического поиска	111
6.3.1	Статистические оконные функции	111
7	Запрос и управление XML-данными	112

7.1	Возвращение результатов в виде XML с помощью предложения FOR XML	112
7.1.1	Режим FOR XML PATH	113
7.2	Запрос XML-данных с помощью XQuery	113
8	Создание таблиц и обеспечение целостности данных	115
8.1	Создание и изменение таблиц	115
8.1.1	Создание таблицы	115
8.1.2	Определение схемы базы данных	115
8.1.3	Свойство идентификатора и порядковые номера	118
8.1.4	Сжатие таблиц	118
8.1.5	Изменение таблицы	119
8.2	Обеспечение целостности данных	122
8.2.1	Использование ограничений	122
9	Проектирование и создание представлений, встроенных функций и синонимов	127
9.1	Проектирование и реализация представлений и встроенных функций	127
9.1.1	Параметры представления	128
9.1.2	Предложение WITH CHECK OPTION	128
9.1.3	Ограничения в представлениях	128
9.1.4	Индексированные представления	129
9.1.5	Выполнение запросов из представлений	129
9.1.6	Изменение представления	130
9.1.7	Удаление представления	130
9.1.8	Модификация данных с помощью представления	130
9.1.9	Секционированные представления	131
9.1.10	Встроенные функции	131
9.1.11	Параметры встроенной функции	133
9.2	Использование синонимов	137
9.2.1	Создание синонима	137
9.2.2	Сравнение синонимов с другими объектами баз данных .	138

10 Вставка, обновление и удаление данных	144
10.1 Вставка данных	144
10.1.1 Инструкция INSERT VALUES	144
10.1.2 Инструкция INSERT SELECT	144
10.1.3 Инструкция INSERT EXEC	145
10.1.4 Инструкция SELECT INTO	145
10.2 Обновление данных	150
10.2.1 Инструкция UPDATE	150
10.2.2 Обновление с использованием объединения	150
10.2.3 Инструкция UPDATE и табличные выражения	150
10.3 Удаление данных	154
10.3.1 Инструкция DELETE	154
10.3.2 Инструкция TRUNCATE	154
10.3.3 Инструкция DELETE на основе объединений	155
10.3.4 Инструкция DELETE с табличными выражениями	156
11 Другие виды модификации данных	162
11.1 Использование объекта последовательности и свойства столбца IDENTITY	162
11.1.1 Использование объекта последовательности	162
11.2 Слияние данных	169
11.2.1 Использование инструкции MERGE	169
11.3 Использование предложения OUTPUT	173
11.3.1 Инструкция INSERT с предложением OUTPUT	173
11.3.2 Инструкция DELETE с предложением OUTPUT	174
11.3.3 Инструкция UPDATE с предложением OUTPUT	174
11.3.4 Инструкция MERGE с предложением OUTPUT	175
11.3.5 Компонуемый DML	175
12 Реализация транзакций, обработка ошибок и динамический SQL	181
12.1 Управление транзакциями и параллелизм	181
12.1.1 Свойства транзакций ACID	181
12.1.2 Типы транзакций	182
12.1.3 Команды транзакций	182

12.1.4 Уровни и состояния транзакций	183
12.1.5 Режимы транзакций	183
12.1.6 Режим автоматической фиксации	183
12.1.7 Режим неявных транзакций	184
12.1.8 Режим явных транзакций	185
12.1.9 Вложенные транзакции	185
12.1.10 Разметка транзакции	186
12.1.11 Дополнительные параметры транзакции	187
12.1.12 Основные блокировки	188
12.1.13 Уровни изоляции транзакций	189
12.2 Реализация обработки ошибок	194
12.2.1 Анализ сообщений об ошибке	194
12.2.2 Команда THROW	195
12.2.3 Неструктурированная обработка ошибок с помощью функции @@ERROR	197
12.2.4 Использование параметра XACT_ABORT с транзакциями	197
12.2.5 Структурированная обработка ошибок с помощью конструкции TRY/CATCH	197
12.3 Использование динамического SQL	202
12.3.1 Инструкция EXECUTE	202
12.3.2 Использование хранимой процедуры sp_executesql	203
13 Разработка и реализация процедур T-SQL	210
13.1 Разработка и реализация хранимых процедур	210
13.1.1 Основные сведения о хранимых процедурах	210
13.1.2 Параметры хранимой процедуры	210
13.1.3 Блок BEGIN/END	211
13.1.4 Инструкция SET NOCOUNT ON	211
13.1.5 Команда RETURN и коды возврата	212
13.1.6 Логика ветвления	212
13.1.7 Конструкция IF/ELSE	213
13.1.8 Конструкция WHILE	213
13.1.9 Команда WAITFOR	213
13.1.10 Команда WAITFOR	214
13.1.11 Вызов других хранимых процедур	214

13.2 Реализация триггеров	218
13.2.1 Триггеры DML	218
13.2.2 Триггеры AFTER	218
13.2.3 Оптимизация триггера	219
13.2.4 Триггеры INSTEAD OF	220
13.2.5 Функции триггеров DML	220
13.3 Основные сведения об определяемых пользователем функциях .	225
13.3.1 Скалярные определяемые пользователем функции	225
13.3.2 Встроенная пользовательская функция с табличным зна- чением	225
13.3.3 Многооператорная пользовательская функция с таблич- ным значением	226
13.3.4 Ограничения для определяемых пользователем функций	226
14 Использование инструментов анализа производительности за- просов	233
14.1 Основные понятия оптимизации запросов	233
14.1.1 Проблемы оптимизации запросов и оптимизатор запросов	233
14.1.2 Подсистема расширенных событий SQL Server, трасси- ровка SQL и приложение SQL Server Profiler	235
14.2 Использование параметров сеанса инструкции SET и анализ планов запросов	238
14.2.1 Параметры сеанса инструкции SET	238
14.2.2 Планы выполнения	239
14.3 Использование динамических административных объектов . .	242
14.3.1 Введение в динамические административные объекты .	242
14.3.2 Наиболее важные динамические административные объ- екты для настройки объектов	243
15 Реализация индексов и статистика	249
15.1 Реализация индексов	249
15.1.1 Кучи	249
15.1.2 Кластеризованные индексы	250
15.1.3 Реализация некластеризованных индексов	251
15.1.4 Реализация индексированных представлений	251

15.2 Использование аргументов поиска	254
15.2.1 Аргументы поиска	254
15.3 Основные понятия статистики	256
15.3.1 Автоматически создаваемая статистика	256
16 Основные сведения о курсорах, наборах данных и временных таблицах	260
16.1 Сравнительная оценка использования решений на основе курсоров/итераций и решений на основе наборов данных	260
16.2 Сравнение использования временных таблиц и табличных переменных	263
17 Основные сведения о дальнейших аспектах оптимизации	268
17.1 Общие сведения об итераторах планов	268
17.2 Использование параметризованных запросов и пакетных операций	271

1 Основы построения запросов

1.1 Основы языка T-SQL

Язык Transact-SQL (T-SQL) — это основной язык, используемый для управления данными и их обработки в Microsoft SQL Server. T-SQL — это диалект стандартного языка SQL.

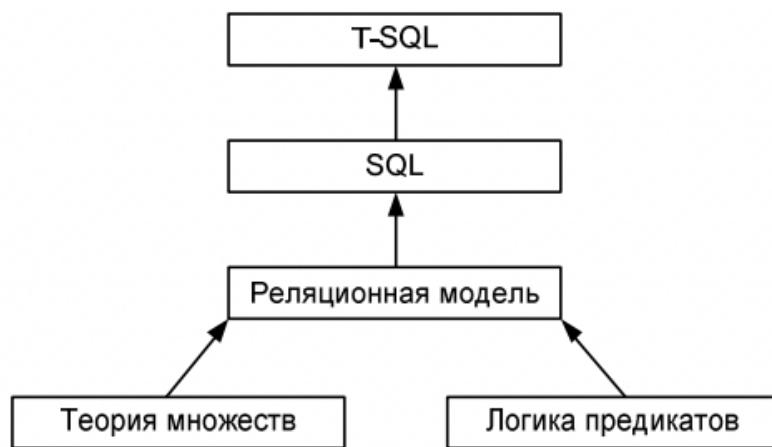


Рис. 1.1: Эволюция языка T-SQL

Следование стандарту при написании кода считается наилучшим решением. Это делает код более переносимым. Если диалект, на котором вы пишете, поддерживает и стандартный, и нестандартный способы выполнения каких-либо операций, всегда следует выбирать стандартный способ. Нестандартный режим стоит выбирать только в том случае, если он дает заметное преимущество, не предоставляемое стандартным режимом.

Основой стандартного языка SQL является *реляционная модель*, представляющая собой математическую модель для управления и обработки данных. Отношение в реляционной модели — это то, что в SQL называется таблицей.

SQL пытается представить отношение с помощью таблицы: отношение имеет заголовок и тело. Заголовок — это набор атрибутов (которые SQL пытается представить в виде столбцов), каждый определенного типа. Атрибут определяется именем и именем типа. Тело отношения представляет собой множество кортежей (которые SQL пытается представить в виде строк).

Каждый заголовок кортежа — это заголовок отношения. Каждое значение атрибута кортежа имеет соответствующий тип.

КОНТРОЛЬНЫЕ ВОПРОСЫ

1. На каких областях математики основывается реляционная модель?
2. В чем заключается разница между T-SQL и SQL?

Ответы на контрольные вопросы

1. Теория множеств и логика предикатов.
2. Язык SQL является стандартом, а T-SQL — это диалект и расширение, реализованное компанией Microsoft в СУБД SQL Server.

Вот пример неправильной терминологии в T-SQL. Термины «поле» (field) и «запись» (record) для обозначения того, что в T-SQL называется «столбец» (column) и «строка» (row) соответственно. Поля и записи — это физические понятия. Поля — это то, что отображается в пользовательском интерфейсе в клиентских приложениях, а записи — это то, что имеется в файлах и курсорах. Таблицы являются логическими структурами и имеют логические строки и столбцы. Еще один пример неверной терминологии связан с «NULL-величинами». NULL — это обозначение отсутствующего значения, а не собственно величины. Поэтому правильное использование этого термина — «NULL-маркер» или просто NULL.

КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Назовите два аспекта, в которых T-SQL отклоняется от реляционной модели.
2. Объясните, как можно обойти эти два аспекта в первом вопросе и использовать T-SQL в реляционной манере.

Ответы на контрольные вопросы

1. Отношение имеет тело с определенным набором кортежей. Таблица не обязательно должна иметь ключ. T-SQL позволяет ссылаться на порядковые позиции столбцов в предложении ORDER BY.
2. Определите ключ в каждой таблице. Ссылайтесь на имена атрибутов, а не на их порядковые позиции в предложении ORDER BY.

КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Почему термины "поле" и "запись" некорректны применительно к столбцу и строке?
2. Почему термин "NULL-значение" некорректен?

Ответы на контрольные вопросы

1. Потому что "поле" и "запись" описывают физические понятия, тогда как столбцы и строки являются логическими элементами таблицы.
2. Потому что NULL не является величиной, напротив, это указание отсутствующего значения.

Резюме занятия

- Язык T-SQL имеет строгие математические основы. Он основывается на стандартном SQL, который, в свою очередь, основывается на реляционной модели, в свою очередь основывающейся на теории множеств и логике предикатов.
- Важно понимать принципы реляционной модели и применять их при написании кода T-SQL.
- При описании концепций в T-SQL необходимо использовать правильную терминологию, поскольку это влияет на ваши знания.

Закрепление материала

1. Почему важно использовать стандартный SQL-код, где это возможно, и знать, какой код является стандартным, а какой — нет? (Выберите все подходящие варианты.)
 - A. Написание кода с использованием стандартного SQL не является важным.
 - B. Стандартный SQL-код более переносим между платформами.
 - C. Стандартный SQL-код является более официальным.
 - D. Знание, что такое стандартный SQL-код, делает ваши знания более платформенно-независимыми.
2. Что из нижеперечисленного не противоречит реляционной модели?
 - A. Использование порядковых позиций для столбцов.
 - B. Возвращение дубликатов строк.
 - C. Не указание ключа в таблице.
 - D. Обеспечение того, чтобы все атрибуты в результате запроса имели имена.
3. Какое взаимоотношение существует между SQL и T-SQL?
 - A. T-SQL является стандартным языком, а SQL — это диалект, используемый в Microsoft SQL Server.
 - B. SQL является стандартным языком, а T-SQL — это диалект, используемый в Microsoft SQL Server.

- C. И SQL, и T-SQL являются стандартными языками.
- D. И SQL, и T-SQL являются диалектами, используемыми в Microsoft SQL Server.

Ответы

Занятие 1

Закрепление материала

1. Правильные ответы: В и D.
 - A. **Неправильно:** следует использовать стандартный код.
 - B. **Правильно:** использование стандартного кода упрощает его перенос между платформами, поскольку требует меньшего количества исправлений.
 - C. **Неправильно:** нет гарантии, что стандартный код будет более эффективным.
 - D. **Правильно:** при использовании стандартного кода проще адаптироваться к новой среде, потому что элементы стандартного кода похожи на разных платформах.
2. Правильный ответ: D.
 - A. **Неправильно:** отношение имеет заголовок с набором атрибутов, и кортежи отношения имеют тот же заголовок. Набор неупорядочен, поэтому порядковые номера не имеют смысла и вступают в противоречие с реляционной моделью. Следует обращаться к атрибутам по их именам.
 - B. **Неправильно:** считается, что запрос должен вернуть отношение. Отношение имеет тело с набором кортежей. Набор не содержит дубликатов. Возвращение дублированных строк является нарушением реляционной модели.
 - C. **Неправильно:** отсутствие ключа в таблице делает возможным наличие дублированных строк в этой таблице, и так же как в предыдущем случае, это противоречит реляционной модели.
 - D. **Правильно:** поскольку ожидается, что атрибуты будут идентифицироваться их именами, гарантия наличия имен у всех атрибутов является реляционной, поэтому несоответствия реляционной модели нет.

644

Ответы

3. Правильный ответ: В.
 - A. **Неправильно:** язык T-SQL не является стандартным, а SQL — это не диалект в Microsoft SQL Server.
 - B. **Правильно:** язык SQL является стандартным, а T-SQL — диалект в Microsoft SQL Server.
 - C. **Неправильно:** T-SQL не является стандартным языком.
 - D. **Неправильно:** SQL это не диалект в Microsoft SQL Server.

1.2 Понимание логической обработки запросов

У языка T-SQL имеется как физическая, так и логическая сторона. Логическая сторона — это концептуальная интерпретация запроса, которая объясняет, что является корректным результатом запроса. Физическая сторона — это обработка запроса ядром базы данных (компонентом Database Engine).

Физическая обработка должна дать результат, определенный логической обработкой запроса. Для достижения этой цели ядро базы данных может использовать оптимизацию. Оптимизация способна изменить последовательность шагов логической обработки запроса или удалить их вовсе, но только при условии, что результат остается тем, который определен логической обработкой запроса.

Логическая последовательность обработки запросов шести основных предложений запросов:

1. FROM.
2. WHERE.
3. GROUP BY.
4. HAVING.
5. SELECT.
6. ORDER BY.

КОНТРОЛЬНЫЙ ВОПРОС

- В чем заключается разница между предложениями WHERE и HAVING?

Ответ на контрольный вопрос

- Предложение WHERE оценивается до группировки строк, и поэтому оно оценивается построчно. Предложение HAVING оценивается после того, как строки сгруппированы, и, следовательно, оценивается для группы.

КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Почему нельзя обращаться к псевдониму столбца, определенному предложением SELECT в предложении WHERE?
2. Почему нельзя обращаться к псевдониму столбца, определенному предложением SELECT в том же самом предложении SELECT?

Ответы на контрольные вопросы

1. Потому что предложение WHERE логически оценивается на этапе, предшествующем этапу, который оценивает предложение SELECT.
2. Потому что все выражения, которые появляются на одном и том же этапе логической обработки запроса, концептуально оцениваются в один и тот же момент времени.

Резюме занятия

- Язык T-SQL разработан как декларативный язык, в котором инструкции представлены в англо-подобном виде. Таким образом, подобный вводу с клавиатуры порядок предложений в запросе начинается с предложения *SELECT*.
- Логическая обработка запросов является концептуальной интерпретацией запроса, определяющей корректный результат, и в отличие от подобного вводу с клавиатуры порядка предложений в запросе, она начинается с оценивания предложения *FROM*.

Закрепление материала

1. Какой из указанных вариантов правильно представляет порядок логической обработки запросов для различных предложений запросов?
 - A. SELECT > FROM > WHERE > GROUP BY > HAVING > ORDER BY.
 - B. FROM > WHERE > GROUP BY > HAVING > SELECT > ORDER BY.
 - C. FROM > WHERE > GROUP BY > HAVING > ORDER BY > SELECT.
 - D. SELECT > ORDER BY > FROM > WHERE > GROUP BY > HAVING.
2. Какой вариант является неправильным? (Выберите все подходящие варианты.)
 - A. Ссылка на атрибут, который сгруппирован в предложении WHERE.
 - B. Ссылка на выражение в предложении GROUP BY; например, GROUP BY YEAR(orderdate).
 - C. В запросе с группировкой ссылка в списке SELECT на атрибут, который не является частью списка GROUP BY и не входит в агрегатную функцию.
 - D. Ссылка на псевдоним, определенный в предложении SELECT, в предложении HAVING.
3. Что справедливо для результата запроса, не содержащего предложение ORDER BY?
 - A. Он является реляционным, если другие требования реляционности соблюdenы.
 - B. Он не может иметь дубликатов.
 - C. Гарантирует ту же последовательность строк в выходных данных, что и последовательность ввода.
 - D. Гарантирует ту же последовательность строк в выходных данных, что и в кластеризованном индексе.

Ответы

Занятие 2

Закрепление материала

1. Правильный ответ: В.
 - A. **Неправильно:** логическая обработка запроса не начинается с предложения `SELECT`.
 - B. **Правильно:** логическая обработка запроса начинается с предложения `FROM` и затем переходит к `WHERE`, `GROUP BY`, `HAVING`, `SELECT` и `ORDER BY`.
 - C. **Неправильно:** предложение `ORDER BY` не оценивается перед предложением `SELECT`.
 - D. **Неправильно:** логическая обработка запроса не начинается с предложения `SELECT`.
2. Правильные ответы: С и D.
 - A. **Неправильно:** T-SQL позволяет обращаться к атрибуту, сгруппированному в предложении `WHERE`.
 - B. **Неправильно:** T-SQL позволяет выполнять группировку по выражению.
 - C. **Правильно:** если в запрос есть группирование, на этапах, обрабатываемых после этапа `GROUP BY`, каждый атрибут, к которому вы обращаетесь, должен присутствовать либо в списке `GROUP BY`, либо в агрегатной функции.
 - D. **Правильно:** поскольку предложение `HAVING` оценивается раньше предложения `SELECT`, ссылка на псевдоним, определенный в предложении `SELECT` внутри предложения `HAVING`, является неправильной.
3. Правильный ответ: А.
 - A. **Правильно:** запрос с предложением `ORDER BY` не возвращает реляционный результат. Чтобы результат был реляционным, запрос должен удовлетворять ряду требований, включая следующие: запрос не должен содержать предложение `ORDER BY`, все атрибуты должны иметь имена, все имена атрибутов должны быть уникальными, и дубликаты не должны присутствовать в результирующем наборе.
 - B. **Неправильно:** запрос, имеющий предложение `DISTINCT` в предложении `SELECT`, может возвратить дубликаты.

- C. **Неправильно:** запрос, не имеющий предложения `ORDER BY`, не может гарантировать последовательность строк на выходе.
- D. **Неправильно:** запрос, не имеющий предложения `ORDER BY`, не может гарантировать последовательность строк на выходе.

Упражнения

Упражнение 1. Важность знания теории

Вы и ваш коллега по команде вступили в дискуссию о важности понимания теоретических основ языка T-SQL. Ваш коллега возражает вам, что понимание этих основ не имеет никакого значения и для того, чтобы быть хорошим разработчиком и писать правильный код, достаточно изучить технические аспекты T-SQL. Ответьте на следующие вопросы, заданные вам вашим коллегой:

1. Можете ли вы привести пример элемента теории множеств, который может улучшить ваше понимание языка T-SQL?
2. Можете ли вы объяснить, почему понимание реляционной модели важно для тех, кто пишет код на языке T-SQL?

Упражнение 2. Собеседование на должность специалиста по анализу кода

Вы проходите собеседование на должность специалиста по анализу кода, чтобы помочь улучшить качество кода. Приложение, с которым работает компания, использует запросы, написанные неквалифицированными специалистами. Эти запросы имеют множество проблем, включая логические дефекты. Ваш интервьюер задал несколько вопросов и хочет получить краткий ответ, состоящий из нескольких фраз, на каждый вопрос. Ответьте на следующие вопросы, заданные вам вашим интервьюером:

1. Важно ли использовать стандартный код, когда это возможно, и почему?
2. У нас есть много запросов, использующих порядковые номера в предложении ORDER BY. Является ли это плохой практикой и, если так, почему?
3. Если в запросе нет предложения ORDER BY, в каком порядке будут возвращены записи?
4. Считаете ли вы правильным использование предложения DISTINCT в каждом запросе?

Ответы

Упражнение 1. Важность знания теории

1. Одна из самых распространенных ошибок, которую делают разработчики T-SQL, заключается во мнении, что запрос без предложения `ORDER BY` всегда возвращает данные в определенном порядке, например, как у кластеризованного индекса. Но как понятно из теории множеств, набор не имеет определенного порядка своих элементов, и вы знаете, что не следует делать таких заключений. В SQL единственным способом гарантировать, что строки будут возвращены в определенном порядке, является добавление предложения `ORDER BY`. Это один из множества примеров аспектов T-SQL, которые можно лучше понять, если вы разбираетесь в основах этого языка.
2. Хотя язык T-SQL основывается на реляционной модели, он во многом отличается от нее. Но он дает вам достаточно инструментов, которые, при условии понимания реляционной модели, позволят вам писать в реляционной манере. Следование реляционной модели поможет писать код более правильно. Далее приведено несколько примеров:
 - не следует полагаться на последовательность столбцов или строк;
 - всегда нужно давать имена результирующим столбцам;
 - следует устранять дубликаты, если возможно их появление в результате запроса.

Упражнение 2. Собеседование на должность специалиста по анализу кода

1. Важно использовать стандартный код SQL. В таком случае и сам код, и ваши знания станут более переносимыми. Особенно в случаях, когда используются как стандартные, так и нестандартные формы элементов языка, рекомендуется использовать стандартные формы.
2. Использование порядковых номеров в предложении `ORDER BY` не рекомендуется. С точки зрения реляционности, предполагается обращение к атрибутам по имени, а не по порядковому номеру. Кроме того, что если список `SELECT` будет изменен в будущем и разработчик забудет откорректировать список `ORDER BY` соответственно?
3. Если запрос не имеет предложения `ORDER BY`, не существует гарантии какой-либо определенной последовательности в результате запроса. Порядок вывода следует считать произвольным. Также надо отметить, что интервьюер использовал

неправильный термин "запись" вместо "строки". Возможно, следует упомянуть об этом, поскольку интервьюер мог сделать это, чтобы проверить ваши знания.

4. С чисто реляционной точки зрения, это может быть допустимо и, возможно, даже рекомендовано. Но с практической точки зрения, существует вероятность, что SQL Server попытается удалить дубликаты, даже если их нет, что приведет к лишним затратам. Поэтому рекомендуется добавлять предложение `DISTINCT` только тогда, когда дубликаты возможны в результирующем наборе, но они не должны быть возвращены.

2 Начало работы с инструкцией SELECT

2.1 Использование FROM и SELECT

```
1  SELECT empid, firstname + N' ' + lastname AS fullname  
2  FROM HR.Employees;
```

Листинг 2.1: Пример работы

Дубликаты возможны в результате запроса и вы хотите их удалить, чтобы получить на выходе реляционный результат, этого можно добиться, добавив предложение DISTINCT, как показано в следующем примере:

```
1  SELECT DISTINCT country, region, city  
2  FROM HR.Employees;
```

Листинг 2.2: Пример работы DISTINCT

Существует интересное различие между стандартным языком SQL и T-SQL с точки зрения минимальных требований запроса SELECT. В соответствии со стандартным языком SQL, запрос должен иметь как минимум предложения FROM и SELECT. Напротив, T-SQL поддерживает запрос SELECT, содержащий только предложение SELECT без предложения FROM.

```
1  SELECT 10 AS col1, 'ABC' AS col2;
```

КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Назовите формы присвоения псевдонимов атрибутам в T-SQL.
2. Что такое нерегулярный идентификатор?

Ответы на контрольные вопросы

1. Формы присвоения псевдонимов: <выражение> AS <псевдоним>, <выражение> <псевдоним> и <псевдоним> = <выражение>.
2. Идентификатор, не соответствующий правилам форматирования идентификаторов, например, начинаящийся с цифры, включающий в себя пробел или являющийся зарезервированным символом T-SQL.

Резюме занятия

- Предложение FROM — это первое предложение, которое логически обрабатывается в запросе SELECT. В этом предложении указываются

таблицы, к которым адресован запрос, и табличные операторы. В предложении FROM можно присваивать таблицам псевдонимы с именами по своему выбору и затем использовать псевдоним таблицы в качестве префикса к именам атрибутов.

- С помощью предложения SELECT можно указать выражения, определяющие результирующие атрибуты. Можно присваивать результирующим атрибутам собственные псевдонимы и таким образом получать реляционный результат. Если в результате запроса возможно получение дубликатов, их следует удалить с помощью предложения DISTINCT.
- При использовании регулярных идентификаторов в качестве имен объектов или атрибутов применение разделителей является необязательным. Если используются нерегулярные идентификаторы, разделители обязательны.

Закрепление материала

1. В чем заключается важность назначения псевдонимов атрибутам в T-SQL? (Выберите все подходящие варианты.)
 - A. Возможность назначать атрибутам псевдонимы является всего лишь эстетическим свойством.
 - B. Выражение, полученное в результате вычисления, не имеет имени атрибута, если его не присвоить с использованием псевдонима, и это нереляционный вариант.
 - C. Язык T-SQL требует, чтобы все результирующие атрибуты запроса имели имена.
 - D. С помощью псевдонимов атрибута можно назначить результирующему атрибуту новое имя, если нужно, чтобы оно отличалось от начального имени атрибута.

42

Глава 2

2. Какие предложения, согласно T-SQL, являются обязательными в запросе SELECT?
 - A. Предложения FROM и SELECT.
 - B. Предложения SELECT и WHERE.
 - C. Предложение SELECT.
 - D. Предложения FROM и WHERE.
3. Какие из следующих методов считаются неправильными? (Выберите все подходящие варианты.)
 - A. Присвоение псевдонимов столбцам с помощью предложения AS.
 - B. Присвоение псевдонимов таблицам с помощью предложения AS.
 - C. Не присваивать псевдоним столбцу, если этот столбец является результатом вычисления.
 - D. Использование знака * в инструкции SELECT.

Ответы

Глава 2

Занятие 1

Закрепление материала

1. Правильные ответы: В и D.
 - A. **Неправильно:** создание псевдонимов атрибутов позволяет удовлетворять требования реляционности, поэтому это явно более чем просто эстетическое свойство.
 - B. **Правильно:** Реляционная модель требует, чтобы все атрибуты имели имена.
 - C. **Неправильно:** язык T-SQL позволяет результирующему атрибуту не иметь имени, если выражение основывается на вычислении без псевдонима.
 - D. **Правильно:** с помощью псевдонима можно назначить результирующему атрибуту имя по своему выбору.
2. Правильный ответ: С.
 - A. **Неправильно:** предложения `FROM` и `SELECT` являются обязательными в запросе `SELECT` по стандарту SQL, но не T-SQL.
 - B. **Неправильно:** предложение `WHERE` необязательное в T-SQL.
 - C. **Правильно:** в соответствии с T-SQL только предложение `SELECT` является обязательным.
 - D. **Неправильно:** оба предложения, и `FROM`, и `WHERE`, необязательные в T-SQL.
3. Правильные ответы: С и D.
 - A. **Неправильно:** присвоение псевдонимов столбцам с помощью предложения `AS` является стандартной процедурой и рекомендуется к применению.
 - B. **Неправильно:** присвоение псевдонимов таблицам с помощью предложения `AS` является стандартной процедурой и рекомендуется к применению.
 - C. **Правильно:** неиспользование псевдонима для столбца, который является результатом вычисления, не является реляционным и не рекомендуется к применению.
 - D. **Правильно:** использование звездочки (*) в списке `SELECT` считается плохой практикой.

2.2 Работа с типами данных истроенными функциями

Подробную информацию и технические подробности о типах данных можно найти в электронной документации по SQL Server 2012 в разделе «Типы данных (Transact-SQL)» по адресу Начало работы с инструкцией SELECT [http://msdn.microsoft.com/ru-ru/library/ms187752\(v=SQL.110\).aspx](http://msdn.microsoft.com/ru-ru/library/ms187752(v=SQL.110).aspx). Для получения дополнительных сведений о встроенных функциях см. раздел «Встроенные функции (Transact-SQL)» по адресу [http://msdn.microsoft.com/ru-ru/library/ms174318\(v=SQL.110\).aspx](http://msdn.microsoft.com/ru-ru/library/ms174318(v=SQL.110).aspx).

Данные с плавающей точкой являются приближенными; поэтому не все

значения в диапазоне такого типа данных могут быть представлены точно. (Эти сведения можно найти в электронной документации для SQL Server 2012 в статье «Типы данных float and real Transact-SQL» по адресу <http://msdn.microsoft.com/ruru/library/ms173773.aspx>.

При использовании символьных строк возникает вопрос о выборе обычных типов символьных данных (CHAR, VARCHAR) или типов в кодировке Unicode (NCHAR, NVARCHAR). Первый тип данных использует 1 байт памяти на символ и поддерживает только один язык (на основании параметров сортировки), кроме английского. Последний тип данных использует 2 байта памяти на символ (без сжатия) и поддерживает несколько языков.

Литералы символьных строк в кодировке Unicode разделяются с помощью заглавной буквы N и затем одиночных кавычек, как в случае N'abc'.

«Приоритет типов данных (Transact-SQL)» на странице <http://msdn.microsoft.com/ru-ru/library/ms190309.aspx>.

2.2.1 Выбор типов данных для ключей

Для генерации суррогатных ключей, как правило, используются следующие элементы:

- **Свойство столбца идентификаторов.** Свойство, которое автоматически генерирует ключи в атрибуте числового типа с масштабом 0; т. е. любого целочисленного типа (TINYINT, SMALLINT, INT, BIGINT) или типа NUMERIC/DECIMAL с масштабом 0.
- **Объект последовательности.** Независимый объект в базе данных, из которого можно получить новые объекты последовательности. Так же как и свойство столбца идентификаторов, он поддерживает любой числовой тип данных с масштабом 0. В отличие от него, он не привязан к конкретному столбцу; напротив, как уже было сказано, это независимый объект в базе данных. Также можно запросить новое значение объекта последовательности перед его использованием.
- **Непоследовательные GUID.** Можно генерировать непоследовательные глобальные уникальные идентификаторы для их сохранения в ат-

рибуте типа UNIQUEIDENTIFIER. Для генерации нового GUID вы можете использовать функцию T-SQL NEWID, вызывая его, например, с выражением по умолчанию, прикрепленным к столбцу. Его также можно генерировать где угодно — например, на клиенте, — с помощью программного интерфейса (API), который генерирует новый GUID. Уникальность идентификаторов GUID гарантирована в пространстве и времени.

- **Последовательные GUID.** Можно генерировать последовательные идентификаторы GUID с помощью функции T-SQL NEWSEQUENTIALID.
- **Пользовательские решения.** Если вы не хотите использовать для генерации ключей встроенные инструменты, предлагаемые SQL Server, следует разработать собственное пользовательское решение. В таком случае тип данных для ключа зависит от пользовательского решения.

СОВЕТ**Подготовка к экзамену**

Для успешной сдачи экзамена важно понимание встроенных инструментов, предлагаемых языком T-SQL для генерации суррогатных ключей, таких как объект последовательности, свойство столбца идентификаторов и функции NEWID и NEWSEQUENTIALID, а также их влияния на производительность.

Обратите внимание, если вы решили остановиться на последовательных ключах и к тому же числового типа, можно всегда начинать с наименьших значений выбранного типа данных, чтобы использовать весь диапазон. Например, для типа INT нужно начинать не с 1, а с числа –2 147 483 648.

Полный список функций, а также технические детали и элементы синтаксиса приведены в электронной документации для SQL Server 2012 в разделе «Типы данных и функции даты и времени (Transact-SQL)» по адресу [http://msdn.microsoft.com/ru-ru/library/ms186724\(v=SQL.110\).aspx](http://msdn.microsoft.com/ru-ru/library/ms186724(v=SQL.110).aspx).

С помощью функции FORMAT можно форматировать входное значение на основе строки форматирования и дополнительно указать культуру (язык и региональные параметры) в качестве третьего входного параметра там, где это имеет смысл. [http://msdn.microsoft.com/ru-ru/library/hh213505\(v=sql.110\).aspx](http://msdn.microsoft.com/ru-ru/library/hh213505(v=sql.110).aspx).

2.2.2 Выражение CASE и связанные с ним функции

```
1  SELECT productid, productname, unitprice, discontinued,
2  CASE discontinued
3    WHEN 0 THEN 'No'
4    WHEN 1 THEN 'Yes'
5    ELSE 'Unknown'
6  END AS discontinued_desc
7  FROM Production.Products;
```

Листинг 2.3: Пример работы CASE

2.2.3 Функция COALESCE

Функция COALESCE принимает список выражений на вход и возвращает первое выражение, не равное NULL, или NULL, если все выражения имеют значение NULL. Например, функция COALESCE(NULL, 'x', 'y') возвращает 'x'. В более общем смысле, функция

```
1  COALESCE(<exp1>, <exp2>, ..., <exprn>);
```

аналогична

```
1  CASE
2    WHEN <exp1> IS NOT NULL THEN <exp1>
3    WHEN <exp2> IS NOT NULL THEN <exp2>
4    ...
5    WHEN <exprn> IS NOT NULL THEN <exprn>
6    ELSE NULL
7  END
```

Типичным использованием функции COALESCE является замена значения NULL чемлибо другим. Например, функция COALESCE(region, '') возвращает значение региона, если это не NULL, и пустую строку, если это NULL.

Функции COALESCE и ISNULL могут влиять на производительность при комбинировании наборов; например, при использовании объединений или при фильтрации данных. Рассмотрим пример, в котором имеются две таблицы — T1 и T2, и необходимо объединить их на основе совпадений между T1.col1 и T2.col1. Атрибуты разрешают значения NULL. Обычно сравнение между двумя значениями NULL дают неизвестную величину, и это приводит к тому, что строки отбрасываются. Вы хотите рассматривать два пустых значения как равные. В таком случае лучшее, что можно сделать — использовать функции COALESCE или ISNULL для замены NULL значением, которое точно не может появиться в данных. Например, если атрибуты целочисленные и вы знаете, что имеете только положительные значения в данных (можно даже установить ограничения для этого), можно использовать предикат COALESCE(T1.col1, -1) = COALESCE(T2.col1, -1) или ISNULL(T1.col1, -1) = ISNULL(T2.col1, -1). Проблема в том, что поскольку вы манипулируете атрибутами, которые сравниваете, SQL Server не может обеспечить упорядочение индекса. Это может привести к эффективному использованию доступных индексов. Рекомендуется применять более длинную форму: T1.col1 = T2.col1 OR (T1.col1 IS NULL AND T2.col1 IS NULL). Её SQL Server понимает как просто сравнение, которое рассматривает пустые значения как равные. С помощью этой формы SQL Server может эффективно использовать индексацию.

КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Будете ли вы использовать тип данных FLOAT для представления цены единицы товара?
2. В чем заключается разница между функциями NEWID и NEWSEQUENTIALID?
3. Какая функция возвращает значение текущей даты и времени с типом данных DATETIME2?
4. В чем заключается разница между оператором + и функцией CONCAT при объединении символьных строк?

Ответы на контрольные вопросы

1. Нет, поскольку тип данных FLOAT — это приблизительный тип данных и не может представлять все значения точно.
2. Функция NEWID генерирует значения идентификатора GUID в произвольном порядке, тогда как функция NEWSEQUENTIALID генерирует идентификаторы GUID, которые увеличиваются последовательно.

3. Функция SYSDATETIME.
4. Оператор + по умолчанию выставляет результирующее значение NULL при входном значении NULL, тогда как функция CONCAT воспринимает значения NULL как пустые строки.

Задание 1. Применение конкатенации строк и использование функций даты и времени

В этом задании вы потренируетесь объединять строки и применять функции даты и времени.

1. Откройте SQL Server Management Studio (SSMS) и подключитесь к учебной базе данных TSQL2012.
2. Напишите запрос к таблице HR.Employees, который возвращает ID сотрудника, его полное имя (объедините атрибуты firstname, пробел и lastname) и год рождения (с применением функции к атрибуту birthdate). Далее приведен вариант запроса, решающего эту задачу.

```
SELECT empid,
       firstname + N' ' + lastname AS fullname,
       YEAR(birthdate) AS birthyear
  FROM HR.Employees;
```

Задание 2. Использование дополнительных функций даты и времени

В этом задании вы попрактикуетесь в использовании дополнительных функций даты и времени.

Напишите выражение, которое вычисляет дату последнего дня текущего месяца. Также напишите выражение, которое вычисляет последний день текущего года. Разумеется, есть масса способов решить эту задачу. Далее приведен один из вариантов вычисления последнего дня последнего месяца.

```
SELECT EOMONTH(SYSDATETIME()) AS end_of_current_month;
```

А в следующем примере приведен один из способов вычисления конца текущего года.

```
SELECT DATEFROMPARTS(YEAR(SYSDATETIME()), 12, 31) AS end_of_current_year;
```

С помощью функции `YEAR` можно извлечь текущий год, а затем предоставить текущий год с номером месяца 12 и количеством дней 31 функции `DATEFROMPARTS` для построения последнего дня текущего года.

Задание 3. Использование строковых данных и функций преобразования

В этом задании вы будете тренироваться в использовании строковых данных и функций преобразования.

1. Напишите запрос к таблице `Production.Products`, который возвращает существующий числовой ID продукта, а также ID продукта в формате строки фиксированной длины в 10 знаков с ведущими нулями. Например, для продукта с ID равным 42, нужно возвратить строку '0000000042'. Один из способов решения этой задачи — использовать следующий код:

```
SELECT productid,
       RIGHT(REPLICATE('0', 10) + CAST(productid AS VARCHAR(10)), 10)
       AS str_productid
  FROM Production.Products;
```

2. Используя функцию `REPLICATE`, сгенерируйте строку, состоящую из 10 нулей. Далее объедините символьную форму ID продукта. Затем извлеките 10 самых правых символов из результирующей строки.

Вы можете предложить более простой способ решения той же задачи с помощью новых функций, появившихся в SQL Server 2012? Значительно проще решить эту задачу с помощью функции `FORMAT`, как показано в следующем примере:

```
SELECT productid,
       FORMAT(productid, 'd10') AS str_productid
  FROM Production.Products;
```

Резюме занятия

- Выбор типа данных для атрибутов оказывает исключительно важное влияние на функциональность и производительность кода T-SQL, взаимодействующего с данными — и еще в большей степени это справедливо для атрибутов, используемых в ключах. Поэтому выбору типов данных следует уделять очень большое внимание.
- Язык T-SQL поддерживает множество функций, которые можно использовать для манипулирования данными даты и времени, символьными строковыми данными и другими типами данных. Помните, что язык T-SQL в основном был предназначен для обработки данных, а не для форматирования или подобных задач. Таким образом, в этих областях, как правило, можно получить только базовую поддержку. Подобные задачи, как правило, лучше всего выполнять на клиенте.

- Язык T-SQL предоставляет выражение CASE, которое позволяет возвращать значение с использованием условной логики, а также множество функций, которые можно рассматривать сокращенными вариантами выражения CASE.

Закрепление материала

1. Почему важно использовать соответствующие типы для атрибутов?
 - Потому что тип атрибута позволяет управлять форматированием значений.
 - Потому что тип атрибута ограничивает значения до определенной области поддерживаемых значений.
 - Потому что тип атрибута предотвращает появление дубликатов.
 - Потому что тип атрибута предотвращает появление значений NULL.
2. Какую из перечисленных далее функций вы будете использовать для генерации суррогатных ключей? (Выберите все подходящие ответы.)
 - NEWID.
 - NEWSEQUENTIALID.
 - GETDATE.
 - CURRENT_TIMESTAMP.
3. В чем состоит разница между простым выражением CASE и поисковым выражением CASE?
 - Простое выражение CASE используется, когда модель восстановления базы данных проста, поисковое выражение CASE используется, когда зарегистрировано полное или неполное восстановление базы данных.
 - Простое выражение CASE сравнивает входное выражение с несколькими возможными выражениями в предложениях WHEN, а поисковое выражение CASE использует независимые предикаты в предложении WHEN.
 - Простое выражение CASE можно использовать где угодно в запросе, а поисковое выражение CASE — только в предложении WHERE.
 - Простое выражение CASE можно использовать где угодно в запросе, а поисковое выражение CASE — только в фильтрах запроса (ON, WHERE, HAVING).

Ответы

1. Правильный ответ: В.
 - А. Неправильно:** как правило, форматирование не входит в область ответственности уровня типов данных или данных; за это отвечает уровень представления данных.
 - В. Правильно:** тип данных должен рассматриваться как ограничение, поскольку он ограничивает разрешенные значения.
 - С. Неправильно:** сам по себе тип данных не препятствует появлению дубликатов. Для предупреждения появления дубликатов необходимо использовать первичный ключ или ограничение уникальности.
 - Д. Неправильно:** тип данных не препятствует появлению значений `NULL`. Чтобы достичь этого, следует использовать ограничение `NOT NULL`.
2. Правильные ответы: А и В.
 - А. Правильно:** функция `NEWID` создает индексы GUID в произвольном порядке. Ее стоит применять, когда дополнительный размер памяти не имеет особого значения, а приоритетной является возможность генерировать уникальное значение во времени и пространстве, из любого места и в произвольном порядке.
 - Б. Правильно:** функция `NEWSEQUENTIALID` генерирует идентификаторы GUID в машине в возрастающей последовательности. Она позволяет уменьшить фрагментацию и хорошо работает, когда данные загружаются в одной сессии и количество дисков невелико. Однако следует тщательно рассмотреть возможность использования другого генератора ключей, такого как объект последовательности, причем с меньшим типом данных, где это возможно.
 - С. Неправильно:** нет никакой гарантии, что функция `GETDATE` сгенерирует уникальные значения; поэтому это не лучший вариант генерации ключей.
 - Д. Неправильно:** функция `CURRENT_TIMESTAMP` — это просто стандартная версия функции `GETDATE`, так что она тоже не гарантирует уникальности.
3. Правильный ответ: В.
 - А. Неправильно:** выражения `CASE` ничего не делают с моделью восстановления базы данных.
 - Б. Правильно:** разница между этими формами заключается в том, что простая форма сравнивает выражения, а поисковая форма использует предикаты.
 - С. Неправильно:** оба выражения `CASE` разрешены везде, где разрешены скалярные выражения — в любом месте запроса.
 - Д. Неправильно:** оба выражения `CASE` разрешены везде, где разрешены скалярные выражения — в любом месте запроса.

Упражнения

Упражнение 1. Анализ использования типов данных

Вы приглашены в качестве консультанта, чтобы решить проблемы с производительностью в существующей системе. Изначально система была разработана с помощью SQL Server 2005 и недавно была обновлена до версии SQL Server 2012.

Скорость записи в системе очень низка, но производительность более чем достаточная. Производительность записи не является приоритетной задачей. Но зато высокий приоритет имеет производительность чтения, которая считается неудовлетворительной на данный момент. Одна из главных задач консультации — представить рекомендации, которые смогут помочь увеличить производительность чтения. У вас назначена встреча с представителями заказчика, и они просят ваших рекомендаций в отношении различных возможностей улучшения ситуации. Одна из интересующих их областей — использование типов данных. Вам надо ответить на следующие вопросы заказчика:

1. Мы используем множество атрибутов, представляющих даты, таких как дата заказа, дата выставления счета-фактуры и т. д., и в данный момент мы используем для этого тип данных DATETIME. Рекомендуете ли вы придерживаться существующего типа данных или заменить его другим? Можете дать еще какие-то аналогичные рекомендации?
2. Мы имеем собственное решение для секционирования таблиц, поскольку используем стандартный выпуск SQL Server. Мы также используем суррогатный ключ типа UNIQUEIDENTIFIER с функцией NEWID, вызываемой выражением ограничения по умолчанию в качестве первичного ключа для таблиц. Этот подход выбран потому, что мы не хотим, чтобы возникали конфликты между ключами в разных таблицах. Этот первичный ключ так же используется, как кластеризованный индексный ключ. Можете ли вы дать рекомендации относительно нашего выбора ключей?

Упражнение 2. Анализ использования функций

Та же компания, которая пригласила вас для анализа использования типов данных, просит вас проанализировать использование у нее функций. Вам задают следующий вопрос.

- Наше приложение до сих пор работало с SQL Server, но из-за недавнего слияния с другой компанией мы также вынуждены поддерживать другие платформы баз данных. Какие рекомендации можете вы нам дать с точки зрения использования функций?

Ответы

Упражнения

Упражнение 1. Анализ использования типов данных

1. Тип данных DATETIME использует 8 байт памяти. SQL Server 2012 поддерживает тип данных DATE, который использует 3 байта памяти. Для всех атрибутов, представляющих только дату, рекомендуется перейти на использование типа данных DATE. Чем меньше требования к памяти, тем лучше будет выполняться чтение.

Что касается прочих рекомендаций, общее правило "чем меньше, тем лучше, при условии, что покрываются потребности атрибута в длительной перспективе" подходит с точки зрения производительности чтения данных. Например, если у вас есть описания переменных длин данных, хранящихся в типе данных CHAR или NCHAR, следует рассмотреть переход на тип данных VARCHAR или NVARCHAR соответственно. Также если в данный момент вы используете типы данных в кодировке Unicode, но вам нужно хранить строковые данные только для одного языка — скажем, US English, — рассмотрите использование стандартных символов.

2. Тип данных UNIQUEIDENTIFIER большой — 16 байт. И поскольку это еще и ключ кластеризованного индекса, он копируется во все некластеризованные индексы. Кроме того, из-за произвольной последовательности, в которой функция NEWID генерирует значения, скорее всего, уровень фрагментации индекса велик. Можно рассмотреть (и протестировать!) другой подход — перейти на целочисленный тип и, используя объект последовательности, генерировать ключи, которые не конфликтуют в таблицах. Из-за меньшего размера типа данных, с учетом эффекта умножения для кластеризованных индексов, производительность чтения, вероятно, станет выше. Значения будут расти, и в результате уменьшится фрагментация, что также положительно повлияет на чтение данных.

Упражнение 2. Анализ использования функций

Для улучшения переносимости кода важно использовать стандартный код, когда это возможно, и конечно, это особенно относится к использованию встроенных функций. Например, используйте функцию COALESCE, а не ISNULL, функцию CURRENT_TIMESTAMP, а не GETDATE, и функцию CASE, а не IIF.

3 Фильтрация и сортировка данных

3.1 Фильтрация данных с помощью предикатов

Использование фильтров в запросе имеет значение и с точки зрения производительности. Прежде всего, выполняя фильтрацию строк в запросе (а не на клиенте), мы снижаем нагрузку на сеть. Кроме того, учитывая информацию фильтров в запросе, SQL Server способен оценить возможность использования индексов для эффективного получения данных без полного сканирования таблицы. Важно заметить, что для эффективного использования индекса предикат должен иметь форму, известную как аргумент поиска (search argument, SARG).

Предикат в форме "столбец оператор значение" или "значение оператор столбец" может быть аргументом поиска. Например, такие предикаты, как `col1 = 10` и `col1 > 10`, являются аргументами поиска. Манипулирование фильтруемыми столбцами в большинстве случаев не позволяет предикатам быть аргументами поиска. Примером манипулирования фильтруемым столбцом может быть применение к нему функции, скажем, $F(\text{col1}) = 10$, где F — некая функция.

```
1 SELECT orderid, orderdate, empid
2 FROM Sales.Orders
3 WHERE COALESCE(shippeddate, '19000101') = COALESCE(@dt, '19000101');
```

Проблема заключается в том, что, хотя такой запрос возвращает правильный результат — даже в случае значения NULL на входе — предикат не является аргументом поиска. Это означает, что SQL Server не может эффективно использовать индекс для столбца `shippeddate`. Для того чтобы сделать предикат аргументом поиска, следует избегать манипулирования фильтруемым столбцом и переписать предикат следующим образом:

```
1 SELECT orderid, orderdate, empid
2 FROM Sales.Orders
3 WHERE shippeddate = @dt
4 OR (shippeddate IS NULL AND @dt IS NULL);
```

СОВЕТ**Подготовка к экзамену**

Для успешного прохождения экзамена необходимо хорошо понимать влияние функций COALESCE и ISNULL на производительность.

В другом примере манипулирования фильтруемый столбец входит в выражение, например, $col1 - 1 \leq @n$. Иногда можно переписать предикат в форме, являющейся аргументом поиска, что делает возможным эффективное использование индексирования. Например, последний предикат можно переписать с помощью простого математического выражения $col1 \leq @n + 1$.

3.1.1 Предикат LIKE

Подстановочный знак	Значение	Пример
% (знак процента)	Любая строка, включая пустую	'D%': строка, начинающаяся с D
_ (подчеркивание)	Один символ	'_D%': строка, в которой второй символ D
[<список символов>]	Один символ из списка	'[AC]%' : строка, в которой первый символ А или С
[<диапазон символов>]	Один символ из диапазона	'[0-9]%' : строка, в которой первый символ — цифра
[^<список или диапазон символов>]	Один символ, не входящий в список или диапазон	'[^0-9]%' : строка, в которой первый символ — не цифра

ВАЖНО!**Производительность предиката LIKE**

Когда шаблон `LIKE` начинается с известного префикса, например `col LIKE 'ABC%`', SQL Server в принципе может эффективно использовать индекс для фильтруемого столбца; иными словами, SQL Server способен спокойно выполнять упорядочение по индексу. Если же шаблон начинается с подстановочного знака, например `col LIKE '%ABC%`', SQL Server уже не может полагаться на упорядочение по индексу. Также при поиске строки, которая начинается с известного префикса (скажем, ABC), надо быть уверенным, что используется предикат `LIKE`, как в случае `col LIKE 'ABC%`, поскольку эта форма считается аргументом поиска. Напомним, что применение обработки к фильтруемому столбцу не позволяет предикату быть аргументом поиска. Например, форма `LEFT(col, 3) = 'ABC'` не является аргументом поиска и не даст SQL Server возможности эффективно использовать индекс.

3.1.2 Фильтрация данных даты и времени

Рекомендуется писать запрос подобно следующему:

```
1 SELECT orderid, orderdate, empid, custid  
2 FROM Sales.Orders  
3 WHERE orderdate = '20070212';
```

Листинг 3.1: Пример работы с фильтрацией даты

ПРИМЕЧАНИЕ Сохранение дат в столбце DATETIME

Фильтруемый столбец `orderdate` имеет тип данных `DATETIME`, представляющий и дату, и время. Литерал, указанный в фильтре, имеет только дату. Когда SQL Server преобразует литерал в тип данных фильтруемого столбца, он устанавливает время на полночь, если время не указано. Если надо, чтобы фильтр возвратил все строки с указанной датой, нужно удостовериться, что все значения сохранены со значением времени "полночь".

Еще один важный аспект фильтрации данных даты и времени — постараться, когда это возможно, использовать аргументы поиска. Например, пусть нужно отфильтровать только заказы, размещенные в феврале 2007 года. Можно использовать функции `YEAR` и `MONTH`, как в следующем примере:

```
1 SELECT orderid, orderdate, empid, custid  
2 FROM Sales.Orders  
3 WHERE YEAR(orderdate) = 2007 AND MONTH(orderdate) = 2;
```

Листинг 3.2: Пример работы с фильтрацией даты без аргумента поиска

Однако поскольку в данном случае выполняются манипуляции с фильтруемым столбцом, предикат не может рассматриваться как аргумент поиска и, следовательно, SQL Server не сможет полагаться на упорядочение по

индексу. Следует переписать этот предикат в виде диапазона, как показано в следующем примере:

```
1  SELECT orderid, orderdate, empid, custid
2  FROM Sales.Orders
3  WHERE orderdate >= '20070201' AND orderdate < '20070301';
```

Листинг 3.3: Пример работы с фильтрацией даты с аргументом поиска

КОНТРОЛЬНЫЕ ВОПРОСЫ

1. В чем заключаются преимущества с точки зрения производительности при использовании фильтра WHERE?
2. Какая форма предиката фильтра может опираться на упорядочивание по индексу?

Ответы на контрольные вопросы

1. Нагрузку на сеть можно уменьшить с помощью выполнения фильтрации на сервере баз данных, а не на клиенте, и можно использовать индексы, чтобы избежать полного сканирования задействованных баз данных.
2. Аргумент поиска (SARG).

Резюме занятия

- Используя предложение WHERE, можно выполнять фильтрацию данных с помощью предикатов. Предикаты в языке T-SQL используют трехуровневую логику. Предложение WHERE возвращает случаи, когда предикат принимает значение «истина» и отбрасывает все прочие.
- Фильтрация данных с предложением WHERE позволяет снизить нагрузку на сеть и может потенциально разрешать использование индексации для минимизации ввода-вывода. Для эффективного использования индексов важно представлять предикаты в виде аргументов поиска.

Закрепление материала

Закрепление материала

Для проверки знания материала, изложенного в данном занятии, ответьте на следующие вопросы. Ответы на эти вопросы и пояснения, почему тот или иной ответ верен или неверен, можно найти в *приложении "Ответы" в конце книги*.

1. Что означает термин "троичная логика" в T-SQL?
 - A. Три возможных логических результирующих значения предиката: истина, ложь и NULL.
 - B. Три возможных логических результирующих значения предиката: истина, ложь и неизвестное значение.
 - C. Три возможных логических результирующих значения предиката: 1, 0 и NULL.
 - D. Три возможных логических результирующих значения предиката: -1, 0 и 1.
2. Какие из перечисленных литералов зависят от языка для типа данных DATETIME? (Выберите все подходящие варианты.)
 - A. '2012-02-12'.
 - B. '02/12/2012'.
 - C. '12/02/2012'.
 - D. '20120212'.

3. Какие из перечисленных предикатов являются аргументами поиска? (Выберите все подходящие варианты.)
 - A. DAY(orderdate) = 1.
 - B. companyname LIKE 'A%'.
C. companyname LIKE '%A%'.
D. companyname LIKE '%A'.
E. orderdate >= '20120212' AND orderdate < '20120213'.

Ответы

Глава 3

Занятие 1

Закрепление материала

1. Правильный ответ: В.
 - A. **Неправильно:** NULL не является частью трех возможных логических результатов предиката в T-SQL.

Ответы

649

- B. **Правильно:** троичная логика оперирует значениями "истина", "ложь" и неизвестное значение.
 - C. **Неправильно:** 1, 0 и NULL не являются частью трех возможных логических результатов предиката.
 - D. **Неправильно:** -1, 0 и 1 не являются частью трех возможных логических результатов предиката.
2. Правильные ответы: А, В и С.
 - A. **Правильно:** форма записи '2012-02-12' не зависит от языка для типов данных DATE, DATETIME2 и DATETIMEOFFSET, но зависит от языка для типов данных DATETIME и SMALLDATETIME.
 - B. **Правильно:** форма записи '02/12/2012' зависит от языка.
 - C. **Правильно:** форма записи '12/02/2012' зависит от языка.
 - D. **Неправильно:** форма записи '20120212' не зависит от языка.
 3. Правильные ответы: В и Е.
 - A. **Неправильно:** этот предикат применяет обработку к фильтруемому столбцу, следовательно, это не аргумент поиска.
 - B. **Правильно:** предикат LIKE является аргументом поиска, когда шаблон начинается с известного префикса.
 - C. **Неправильно:** предикат LIKE не является аргументом поиска, когда шаблон начинается с подстановочного знака.
 - D. **Неправильно:** предикат LIKE не является аргументом поиска, когда шаблон начинается с подстановочного знака.
 - E. **Правильно:** предикат является аргументом поиска, поскольку к фильтруемому столбцу не применяется никаких манипуляций.

3.2 Сортировка данных

КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Как можно гарантировать последовательность строк в результате запроса?
2. В чем состоит разница между результатом запроса с предложением ORDER BY и без него?

Ответы на контрольные вопросы

1. Единственный способ — добавить предложение ORDER BY.
2. Без предложения ORDER BY результат будет реляционным (с точки зрения сортировки); при наличии предложения ORDER BY результат в принципе представляет собой то, что стандартно называется курсором.

Резюме занятия

- Запросы, как правило, возвращают реляционный результат там, где сортировка не гарантирована. Если вам необходимо гарантировать упорядочение представления, нужно в запросе добавить предложение ORDER BY для манипулирования данными даты и времени, символьными строковыми данными и другими типами данных. Помните, что язык T-SQL в основном был предназначен для обработки данных, а не для форматирования или подобных задач. Таким образом, в этих областях, как правило, можно получить только базовую поддержку. Подобные задачи, как правило, лучше всего выполнять на клиенте.
- С помощью предложения ORDER BY можно указать список выражений для первичной сортировки, вторичной сортировки и т. д. Для каждого выражения можно указать параметры ASC и DESC для упорядочения по возрастанию или убыванию, при этом по умолчанию принимается сортировка по возрастанию.

- Даже если указано предложение ORDER BY, в результате все равно может получиться недетерминированная сортировка. Чтобы она была детерминированной, список ORDER BY должен быть уникальным.
- Можно использовать порядковые номера выражений из списка SELECT в предложении ORDER BY, но это считается плохой практикой.
- Можно выполнять сортировку по элементам, появляющимся в списке SELECT, если предложение DISTINCT также не определено.
- Поскольку считается, что предложение ORDER BY обрабатывается после предложения SELECT, можно ссылаться на псевдонимы, присвоенные в предложении SELECT внутри предложения ORDER BY.
- При сортировке SQL Server считает значения NULL ниже, чем значения не-NUL, и равными друг другу. Это означает, что при упорядочивании по возрастанию они сортируются все вместе перед не-NULL-маркерами.

Закрепление материала

1. Если в запросе отсутствует предложение ORDER BY, в каком порядке будутозвращены строки?
 - A. В произвольном порядке.
 - B. С сортировкой по первичному ключу.

90

Глава 3

- C. С сортировкой по кластерному индексу.
 - D. В порядке размещения.
2. Вы хотите, чтобы результирующие строки были отсортированы по убыванию значения orderdate и затем по убыванию значения orderid. Какое из приведенных далее предложений даст желаемый результат?
 - A. ORDER BY orderdate, orderid DESC.
 - B. ORDER BY DESC orderdate, DESC orderid.
 - C. ORDER BY orderdate DESC, orderid DESC.
 - D. DESC ORDER BY orderdate, orderid.
 3. Вы хотите, чтобы результирующие строки были отсортированы по возрастанию значения orderdate и затем по возрастанию значения orderid. Какие из приведенных далее предложений дадут желаемый результат? (Укажите все возможные варианты.)
 - A. ORDER BY ASC(orderdate, orderid).
 - B. ORDER BY orderdate, orderid ASC.
 - C. ORDER BY orderdate ASC, orderid ASC.
 - D. ORDER BY orderdate, orderid.

Ответы

Занятие 2

Закрепление материала

1. Правильный ответ: А.
 - A. **Правильно:** без предложения ORDER BY сортировка не гарантирована и может быть произвольной — это зависит от оптимизации.
 - B. **Неправильно:** без предложения ORDER BY сортировка не гарантирована.
 - C. **Неправильно:** без предложения ORDER BY сортировка не гарантирована.
 - D. **Неправильно:** без предложения ORDER BY сортировка не гарантирована.
2. Правильный ответ: С.
 - A. **Неправильно:** здесь используется сортировка по возрастанию для orderdate и по убыванию для orderid.
 - B. **Неправильно:** это неверный синтаксис.

650

Ответы

- C. **Правильно:** правильный синтаксис — указать DESC после каждого выражения, для которого направление сортировки должно быть по убыванию.
- D. **Неправильно:** это неверный синтаксис.
3. Правильные ответы: В, С и D.
 - A. **Неправильно:** это неверный синтаксис.
 - B. **Правильно:** по умолчанию выполняется сортировка по возрастанию, поэтому данное предложение использует сортировку по возрастанию и для orderdate, и для orderid.
 - C. **Правильно:** это предложение явно использует сортировку по возрастанию и для orderdate, и для orderid.
 - D. **Правильно:** по умолчанию выполняется сортировка по возрастанию, поэтому данное предложение использует сортировку по возрастанию и для orderdate, и для orderid.

3.3 Фильтрация данных с помощью ТОР и OFFSET...FETCH

3.3.1 Фильтрация данных с помощью предложения ТОР

```
1 SELECT TOP (3) orderid, orderdate, custid, empid  
2 FROM Sales.Orders  
3 ORDER BY orderdate DESC;
```

Листинг 3.4: Пример работы с ТОР

ПРИМЕЧАНИЕ Параметр ТОР и круглые скобки

Язык T-SQL поддерживает указание числа строк, которое надо отфильтровать, с помощью параметра ТОР в запросах SELECT без использования скобок, но только лишь с целью обратной совместимости. Правильный синтаксис предполагает использование круглых скобок.

Вместо числа строк можно также указать процентное соотношение строк, которые надо отфильтровать. Для этого укажите величину FLOAT в диапазоне от 0 до 100 в скобках и ключевое слово PERCENT после скобок, как показано в следующем примере:

```
1 SELECT TOP (1) PERCENT orderid, orderdate, custid, empid  
2 FROM Sales.Orders  
3 ORDER BY orderdate DESC;
```

Параметр PERCENT задает следующую целую часть результирующего количества строк, если это не целое число. В нашем примере без использования предложения ТОР количество строк в результирующем наборе равно 830. Фильтрация 18,3, и следующая целая часть этого числа равна 9; следовательно, этот запрос возвратит 9 строк.

Предложение ТОР не ограничено постоянным вводом, наоборот, оно позволяет указывать произвольное выражение. С практической точки зрения эта возможность особенно важна, когда необходимо передать параметр переменной в качестве входных данных, как в приведенном далее примере кода.

```
1  DECLARE @n AS BIGINT = 5;
2  SELECT TOP (@n) orderid, orderdate, custid, empid
3  FROM Sales.Orders
4  ORDER BY orderdate DESC;
```

Однако этот запрос не является детерминированным. Он фильтрует 3 строки, но без всякой гарантии, какие именно три строки будут возвращены. Вы в итоге получите любые 3 строки, которые SQL Server выбрал первыми, и это зависит от оптимизации.

Мы не всегда заботимся о том, чтобы результат был детерминированным или повторяемым, но если это необходимо, можно использовать две возможности. Одна — попросить включить все связи с последней строкой, добавив аргумент WITH TIES, как на примере далее.

```
1  SELECT TOP (3) WITH TIES orderid, orderdate, custid, empid
2  FROM Sales.Orders
3  ORDER BY orderdate DESC;
```

3.3.2 Фильтрация данных с помощью OFFSET...FETCH

Следующий запрос задает сортировку на основе даты заказа по убыванию, далее — идентификатора заказа по убыванию, а затем он пропускает 50 строк и выбирает следующие 25 строк.

```
1  SELECT orderid, orderdate, custid, empid
2  FROM Sales.Orders
3  ORDER BY orderdate DESC, orderid DESC
4  OFFSET 50 ROWS FETCH NEXT 25 ROWS ONLY;
```

С помощью предложений OFFSET и FETCH можно в качестве входных данных использовать выражения. Это очень удобно, когда требуется динамически вычислять входные значения. Например, представьте, что вы реализуете возможность постраничного просмотра, где пользователю возвращается одна страница строк за один раз. Пользователь отправляет вашей процедуре или функции в качестве входных параметров номер страницы (@pagenum

parameter) и размер страницы (@pagesize parameter). Это означает, что вам нужно пропустить количество строк, равное @pagenum минус 1, умноженное на @pagesize, и выбрать следующие @pagesize строк. Реализовать это можно с помощью следующего кода (с использованием локальных переменных для простоты):

```
1  DECLARE @pagesize AS BIGINT = 25, @pagenum AS BIGINT = 3;
2  SELECT orderid, orderdate, custid, empid
3  FROM Sales.Orders
4  ORDER BY orderdate DESC, orderid DESC
5  OFFSET (@pagenum - 1) * @pagesize ROWS FETCH NEXT @pagesize ROWS ONLY;
```

Поскольку конструкция OFFSET...FETCH является стандартной, а TOP — нет, в случаях, когда они логически эквивалентны, рекомендуется использовать более привычный. Кроме того, OFFSET...FETCH имеет преимущество перед параметром TOP — он поддерживает возможность пропуска данных. Однако OFFSET...FETCH не поддерживает имеющиеся у TOP возможности, такие как PERCENT и WITH TIES.

КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Как гарантировать детерминированные результаты с помощью конструкции TOP?
2. Каковы преимущества использования OFFSET...FETCH по сравнению с TOP?

Ответы на контрольные вопросы

1. Либо возвращая все связи с помощью параметра WITH TIES, либо с использованием уникальной сортировки для разрыва связей.
2. Конструкция OFFSET...FETCH является стандартной, тогда как TOP — нет; кроме того, OFFSET...FETCH поддерживает возможность пропуска данных, а TOP — нет.

Практикум

ПРАКТИКУМ Фильтрация данных с помощью *TOP* и *OFFSET...FETCH*

В этом практикуме вам предстоит проверить ваши знания о фильтрации данных с помощью конструкций *TOP* и *OFFSET...FETCH*.

Задание 1. Использование конструкции *TOP*

В этом задании вы будете использовать конструкцию *TOP* для фильтрации данных.

1. Откройте SQL Server Management Studio (SSMS) и подключитесь к учебной базе данных TSQL2012.
2. Перед вами поставлена задача написать запрос к таблице *Production.Products*, возвращающий 5 наиболее дорогих продуктов 1-й категории. Напишите следующий запрос:

```
SELECT TOP (5) productid, unitprice
FROM Production.Products
WHERE categoryid = 1
ORDER BY unitprice DESC;
```

Вы получите следующий результирующий набор:

Productid	unitprice
38	263.50
43	46.00
2	19.00
1	18.00
35	18.00

Запрос возвращает нужный результат, за исключением того, что нет никакой обработки связей. Иными словами, упорядочение продуктов с одинаковой ценой не является детерминированным.

3. Вас просят представить решения для превращения предыдущего запроса в детерминированный: одно решение с использованием связей, другое — с разрывом связей. Во-первых, рассмотрите вариант, который включает все связи, с помощью параметра *WITH TIES*. Добавьте этот параметр следующим образом.

```
SELECT TOP (5) WITH TIES productid, unitprice
FROM Production.Products
WHERE categoryid = 1
ORDER BY unitprice DESC;
```

Вы получите следующий результат, включающий связи.

productid	unitprice
38	263.50
43	46.00
2	19.00
1	18.00
39	18.00
35	18.00
76	18.00

4. Используйте следующий вариант, который разрывает связи, используя параметр `productid` в убывающем порядке.

```
SELECT TOP (5) productid, unitprice
FROM Production.Products
WHERE categoryid = 1
ORDER BY unitprice DESC, productid DESC;
```

Этот запрос дает следующий результат:

productid	unitprice
38	263.50
43	46.00
2	19.00
76	18.00
39	18.00

Задание 2. Использование конструкции `OFFSET...FETCH`

В этом задании вы попробуете свои силы в использовании конструкции `OFFSET...FETCH` для фильтрации данных.

1. Откройте SQL Server Management Studio (SSMS) и подключитесь к учебной базе данных TSQL2012.
2. Вам необходимо написать несколько запросов, которые просматривают продукты, по 5 за один раз, с сортировкой по цене единицы товара, используя идентификатор продукта (`productid`) для разрыва соединений. Начните с написания запроса, который возвращает пять первых продуктов.

```
SELECT productid, categoryid, unitprice
FROM Production.Products
ORDER BY unitprice, productid
OFFSET 0 ROWS FETCH FIRST 5 ROWS ONLY;
```

Вы могли бы использовать либо ключевое слово FIRST, либо NEXT, предположим, что вы решили использовать ключевое слово FIRST, поскольку это более естественно, если не надо пропускать строки. Этот запрос генерирует следующий результат:

productid	categoryid	unitprice
33	4	2.50
24	1	4.50
13	8	6.00
52	5	7.00
54	6	7.45

3. Далее, напишите запрос, который возвращает следующие 5 строк (с 6 по 10), используя следующий пример:

```
SELECT productid, categoryid, unitprice
FROM Production.Products
ORDER BY unitprice, productid
OFFSET 5 ROWS FETCH NEXT 5 ROWS ONLY;
```

На этот раз используйте ключевое слово NEXT, поскольку вам нужно пропустить несколько строк. Этот запрос генерирует следующий результат:

productid	categoryid	unitprice
75	1	7.75
23	5	9.00
19	3	9.20
45	8	9.50
47	3	9.50

4. Аналогично, напишите запрос, который возвращает строки с 11 по 15:

```
SELECT productid, categoryid, unitprice
FROM Production.Products
ORDER BY unitprice, productid
OFFSET 10 ROWS FETCH NEXT 5 ROWS ONLY;
```

Этот запрос генерирует следующий результат:

productid	categoryid	unitprice
41	8	9.65
3	2	10.00
21	3	10.00
74	7	10.00
46	8	12.00

То же самое надо сделать для последующих строк.

Резюме занятия

- С помощью конструкций TOP и OFFSET...FETCH можно фильтровать даты на основе указанного количества строк и сортировки.
- Предложение ORDER BY, которое обычно используется в запросе для сортировки представления, также используется конструкциями TOP и OFFSET...FETCH, чтобы указать, какие строки надо фильтровать.
- Параметр TOP — собственная возможность языка T-SQL, которую можно использовать для указания количества или процентного соотношения строк, подлежащих фильтрации.
- Можно сделать запрос TOP детерминированным двумя способами: первый — используя параметр WITH TIES для возвращения всех связей, второй — используя уникальную сортировку для разрыва связей.
- OFFSET...FETCH — это стандартная конструкция, подобная параметру TOP, поддерживаемая SQL Server 2012. В отличие от TOP она позволяет задать количество строк, которые надо пропустить, прежде чем указать число строк, которое следует отфильтровать. Поэтому она может использоваться для оперативной разбивки данных на страницы.
- Как TOP, так и OFFSET...FETCH поддерживают в качестве входных данных выражения, а не только константы.

Закрепление материала

Закрепление материала

Для проверки знания материала, изложенного в данном занятии, ответьте на следующие вопросы. Ответы на эти вопросы и пояснения, почему тот или иной ответ верен или неверен, можно найти в *приложении "Ответы" в конце книги*.

1. Вы выполняете запрос с выражением `TOP (3)`. Какой из следующих вариантов наиболее точно описывает, сколько строк будет возвращено?
 - A. Меньше трех строк.
 - B. Три строки или менее.
 - C. Три строки.
 - D. Три строки или более.
 - E. Более трех строк.
 - F. Меньше трех, три или более трех строк.
2. Вы выполняете запрос с выражением `TOP (3) WITH TIES` и неуникальной сортировкой. Какой из следующих вариантов наиболее точно описывает, сколько строк будет возвращено?
 - A. Меньше трех строк.
 - B. Три строки или менее.
 - C. Три строки.

- D. Три строки или более.
 - E. Более трех строк.
 - F. Меньше трех, три или более трех строк.
3. Какое из следующих `OFFSET...FETCH` выражений корректно в T-SQL? (Укажите все подходящие варианты.)
 - A. `SELECT ... ORDER BY orderid OFFSET 25 ROWS.`
 - B. `SELECT ... ORDER BY orderid FETCH NEXT 25 ROWS ONLY.`
 - C. `SELECT ... ORDER BY orderid OFFSET 25 ROWS FETCH NEXT 25 ROWS ONLY.`
 - D. `SELECT ... <no ORDER BY> OFFSET 0 ROWS FETCH FIRST 25 ROWS ONLY.`

Ответы

-
1. Правильный ответ: B.
 - A. **Неправильно:** если в результате запроса есть хотя бы три строки, не содержащие `TOP`, запрос возвратит три строки.
 - B. **Правильно:** если в результате запроса меньше трех строк, не содержащих `TOP`, запрос возвратит эти строки. Если таких строк три или более, запрос возвратит три строки.
 - C. **Неправильно:** если в результате запроса меньше трех строк, не содержащих `TOP`, запрос возвратит эти строки.
 - D. **Неправильно:** если не будет использоваться опция `WITH TIES`, запрос не будет возвращать больше, чем требуемое число строк.
 - E. **Неправильно:** если не будет использоваться опция `WITH TIES`, запрос не будет возвращать больше, чем требуемое число строк.
 - F. **Неправильно:** если не будет использоваться опция `WITH TIES`, запрос не будет возвращать больше, чем требуемое число строк.
 2. Правильный ответ: F.
 - A. **Неправильно:** если в результате запроса есть хотя бы три строки, не содержащие `TOP`, запрос возвратит хотя бы три строки.
 - B. **Неправильно:** если в результате запроса более трех строк, не содержащих `TOP`, и имеются связи с третьей строкой, запрос возвратит более трех строк.
 - C. **Неправильно:** если в результате запроса меньше трех строк, не содержащих `TOP`, запрос возвратит только эти строки. Если в результате запроса более трех строк, не содержащих `TOP`, и имеются связи с третьей строкой, запрос возвратит более трех строк.

- D. **Неправильно:** если в результате запроса меньше трех строк, не содержащих `TOP`, запрос возвратит только эти строки.
 - E. **Неправильно:** если в результате запроса три или меньше строк, не содержащих `TOP`, запрос не возвратит более трех строк.
 - F. **Правильно:** если в результате запроса меньше трех строк, не содержащих `TOP`, запрос возвратит только эти строки. Если в результате запроса есть хотя бы три строки и не существует связей с третьей строкой, запрос возвратит три строки. Если в результате запроса более трех строк и имеются связи с третьей строкой, запрос возвратит более трех строк.
3. Правильные ответы: А и С.
 - A. **Правильно:** T-SQL поддерживает использование предложения `OFFSET` без предложения `FETCH`.
 - B. **Неправильно:** в отличие от стандартного SQL, язык T-SQL не поддерживает предложение `FETCH` без предложения `OFFSET`.
 - C. **Правильно:** T-SQL поддерживает использование предложений `OFFSET` и `FETCH`.
 - D. **Неправильно:** T-SQL не поддерживает `OFFSET...FETCH` без предложения `ORDER BY`.

Упражнения

Упражнение 1. Рекомендации по улучшению производительности фильтрации и сортировки

Вы приглашены в качестве консультанта на пивоваренный завод, использующий SQL Server 2012, чтобы помочь решить проблемы с производительностью. Вы выполняете трассировку обычной нагрузки на систему и обнаруживаете очень медленное выполнение запросов. Вы видите очень большую нагрузку на сеть. Вы видите, что множество запросов возвращает все строки клиенту и затем клиент выполняет фильтрацию. Запросы, которые фильтруют данные, часто выполняют обработку фильтруемых столбцов. Все запросы содержат предложение ORDER BY, и когда вы об этом спрашиваете, вам говорят, что в действительности это не требуется, но разработчики привыкли так делать — на всякий случай. Вы обнаружили множество дорогостоящих операций сортировки. Клиент хочет получить от вас рекомендации по улучшению производительности и задает вам следующие вопросы:

1. Можно ли сделать что-нибудь, чтобы улучшить способ выполнения сортировки?
2. Наносит ли какой-либо ущерб использование ORDER BY, даже если данные не обязательно должны возвращаться в отсортированном виде?
3. Дайте, пожалуйста, рекомендации по использованию запросов с конструкциями TOP и OFFSET...FETCH?

Упражнение 2. Обучение разработчика-стажера

Вы обучаете разработчика-стажера фильтрации и сортировке данных на языке T-SQL. Разработчику не все понятно, и он задает вам вопросы. Ответьте на следующие вопросы, используя все свои знания:

1. Когда я пытаюсь сослаться на псевдоним столбца, который определил в списке SELECT в предложении WHERE, то получаю ошибку. Объясните, пожалуйста, почему это запрещено и как разрешить эту проблему?
2. Кажется, ссылка на псевдоним столбца в предложении ORDER BY поддерживается. Почему?
3. Почему так получилось, что компания Microsoft сделала обязательным указывать предложение ORDER BY при использовании конструкции OFFSET...FETCH, но не при использовании TOP? Означает ли это, что только запросы TOP могут иметь недетерминированную сортировку?

Ответы

Упражнение 1. Рекомендации по улучшению производительности фильтрации и сортировки

1. Прежде всего, в базе данных должно выполняться как можно больше фильтрации данных. Выполнение большей части фильтрации данных на клиенте означает, что вы сканируете больше данных, что увеличивает нагрузку на подсистему хранения данных, а также что вы увеличиваете нагрузку на сеть. При выполнении фильтрации в базе данных, например, с помощью предложения `WHERE`, следует использовать аргументы поиска, которые повышают вероятность эффективного применения индексов. Следует, насколько возможно, избегать манипулирования фильтруемыми столбцами.
2. Добавление предложения `ORDER BY` означает, что SQL Server должен гарантировать возвращение строк в запрашиваемой последовательности. Если нет существующих индексов для поддержки требований сортировки, SQL Server не будет иметь другой возможности, кроме как сортировать данные. На больших наборах сортировка является дорогой. Поэтому главная рекомендация — избегать добавления предложений `ORDER BY` в запросы, если нет требований сортировки данных. И когда действительно нужно возвращать строки в определенном порядке, следует рассмотреть возможность организации поддерживающих индексов, чтобы избавить SQL Server от необходимости выполнять дорогостоящие операции сортировки.

652

Ответы

3. Главный способ помочь хорошей работе запросов, содержащих `TOP` и `OFFSET...FETCH`, — организация индексов для поддержки элементов сортировки. Это, в дополнение к сортировке, может предотвратить сканирование всех данных.

Упражнение 2. Обучение разработчика-стажера

1. Чтобы понять, почему нельзя ссылаться на псевдоним, определенный в списке `SELECT` в предложении `WHERE`, надо понимать логическую обработку запросов. Несмотря на то, что последовательность ввода предложений — `SELECT...FROM...WHERE...GROUP BY...HAVING...ORDER BY`, последовательность логической обработки запросов `FROM...WHERE...GROUP BY...HAVING...SELECT...ORDER BY`. Как видите, предложение `WHERE` оценивается раньше предложения `SELECT`, и таким образом, псевдонимы, определенные в предложении `SELECT`, не видны предложению `WHERE`.
2. Последовательность логической обработки запросов объясняет, почему предложение `ORDER BY` может ссылаться на псевдонимы, определенные в предложении `SELECT`. Это происходит потому, что предложение `ORDER BY` логически оценивается после предложения `SELECT`.
3. Предложение `ORDER BY` является обязательным, если используется `OFFSET...FETCH`, поскольку это стандартное предложение, и стандартный SQL решил сделать его обязательным. Компания Microsoft просто следуя стандарту. Что касается конструкции `TOP`, это частное свойство, и когда компания Microsoft разрабатывала его, ее сотрудники решили разрешить использование `TOP` в совершенно недетерминированной манере — без предложения `ORDER BY`. Обратите внимание, то, что `OFFSET...FETCH` требует наличия предложения `ORDER BY`, не означает, что вы должны использовать детерминированную сортировку. Например, если список `ORDER BY` не является уникальным, сортировка не будет детерминированной. И если нужно, чтобы сортировка полностью была недетерминированной, можно указать выражение `ORDER BY (SELECT NULL)`, и это эквивалентно тому, что предложение `ORDER BY` не указано совсем.

4 Комбинирование наборов данных

4.1 Использование соединений

4.1.1 Перекрестные соединения CROSS JOIN

Это соединение выполняет то, что называют декартовым произведением двух входных таблиц

```
1 SELECT D.n AS theday, S.n AS shiftno
2 FROM dbo.Nums AS D
3     CROSS JOIN dbo.Nums AS S
4 WHERE D.n <= 7
5     AND S.N <= 3
6 ORDER BY theday, shiftno;
```

Листинг 4.1: Пример CROSS JOIN

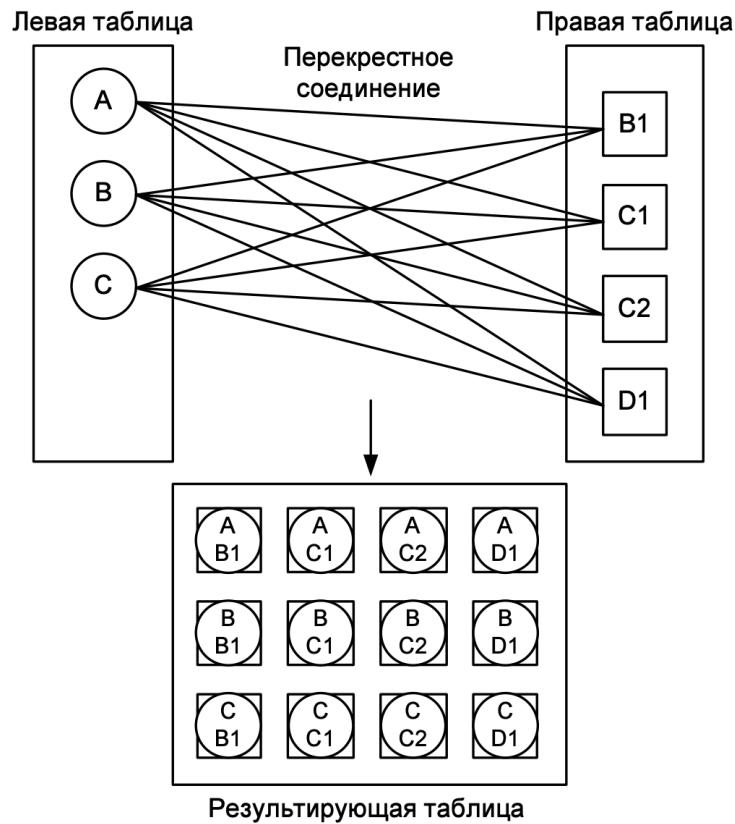


Рис. 4.1: Перекрестное объединение таблиц

4.1.2 Внутренние соединения INNER JOIN

С помощью внутренних соединений можно сопоставлять строки из двух таблиц по предикату, как правило, сравнивая значение первичного ключа в одной таблице с внешним ключом в другой.

```
1  SELECT S.companyname AS supplier, S.country,
2      P.productid, P.productname, P.unitprice
3  FROM Production.Suppliers AS S
4  INNER JOIN Production.Products AS P
5      ON S.supplierid = P.supplierid
6  WHERE S.country = N'Japan';
```

Листинг 4.2: Пример INNER JOIN

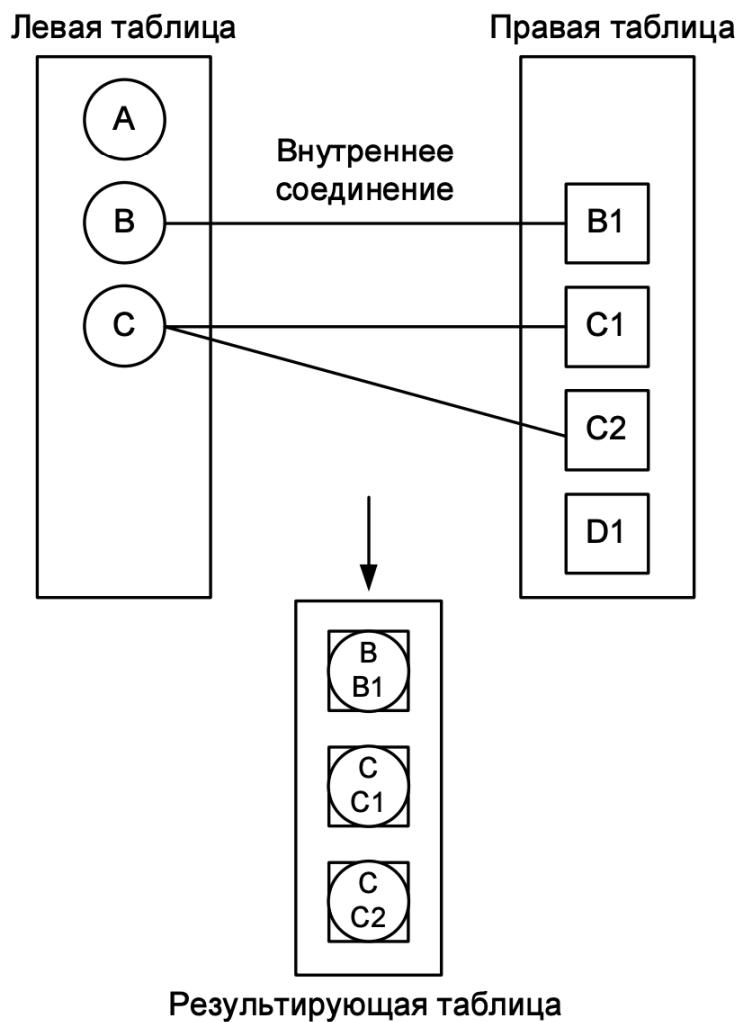


Рис. 4.2: Внутреннее объединение таблиц

Часто таблицы объединяют по внешнему ключу. Например, имеется внешний ключ, определенный для столбца `supplierid` в таблице `Production.Products` (ссылающаяся таблица). Также важно помнить, что когда определяется первичный ключ или уникальная константа, SQL Server создает уникальный индекс для столбцов ограничения, чтобы обеспечить свойство уникальности ограничения. Но при определении внешнего ключа SQL Server не создает никаких индексов на столбцах внешнего ключа. Такие индексы способны улучшить производительность соединений по их отношениям. Поскольку SQL Server не создает такие индексы автоматически, вы должны самостоятельно определить, где они могут быть полезны, и создать их. Таким образом, при настройке индексов следует проанализировать столбцы внешнего ключа и оценить преимущества создания на них индексов.

4.1.3 Внешние соединения OUTER JOIN

С помощью внешнего соединения можно запросить сохранение всех строк из одной или обеих сторон соединения без учета того, имеются ли соответствующие строки в другой стороне. Это осуществляется с помощью предиката `ON`.

```
1 SELECT S.companyname AS supplier, S.country,
2     P.productid, P.productname, P.unitprice
3 FROM Production.Suppliers AS S
4 LEFT OUTER JOIN Production.Products AS P
5     ON S.supplierid = P.supplierid
6 WHERE S.country = N'Japan';
```

Листинг 4.3: Пример LEFT JOIN

Важно понимать, что в случае внешнего соединения предложения `ON` и `WHERE` играют разные роли и поэтому не являются взаимозаменяемыми. Предложение `WHERE` по-прежнему играет роль фильтра — а именно, оно сохраняет строки, дающие значение "истина" и отбрасывает строки, дающие значение "ложь" или "неизвестно". Так, в нашем запросе предложение `WHERE` фильтрует только поставщиков из Японии, поэтому поставщики из других стран просто не попадают в выходной набор. Однако предложение `ON` не является просто фильтром, напротив, его основная задача — сопоставление данных. Иными словами, строка из сохраненной стороны будет возвращена независимо от того, найдет предикат `ON` совпадение или нет.

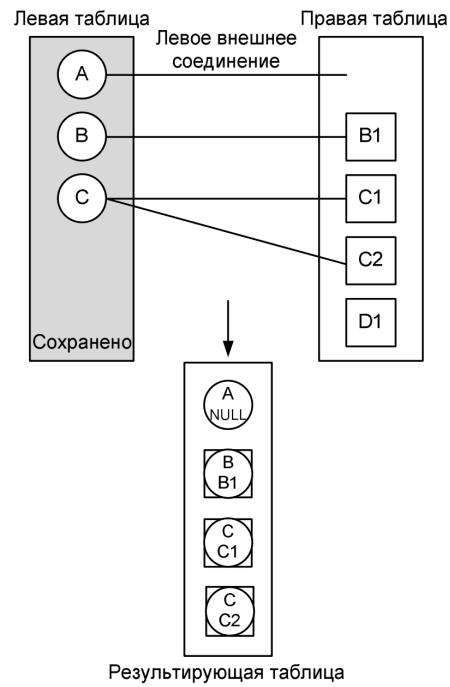


Рис. 4.3: Левое внешнее соединение

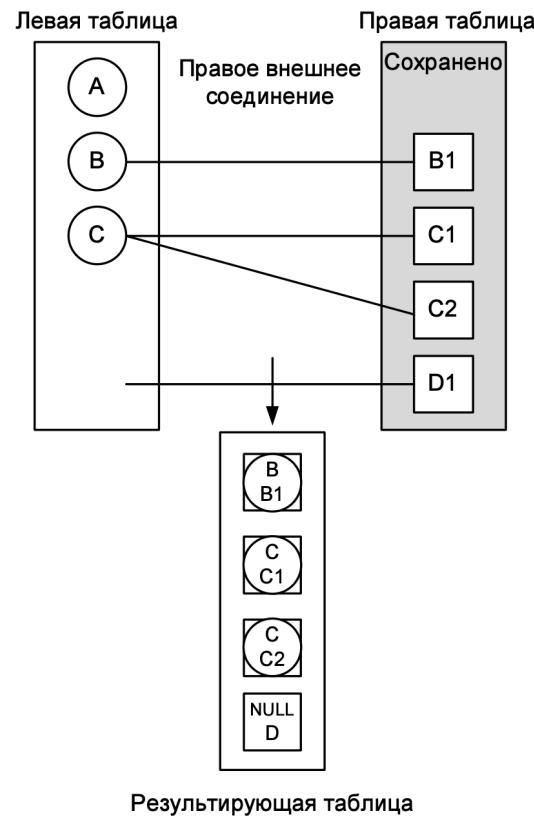


Рис. 4.4: Правое внешнее соединение

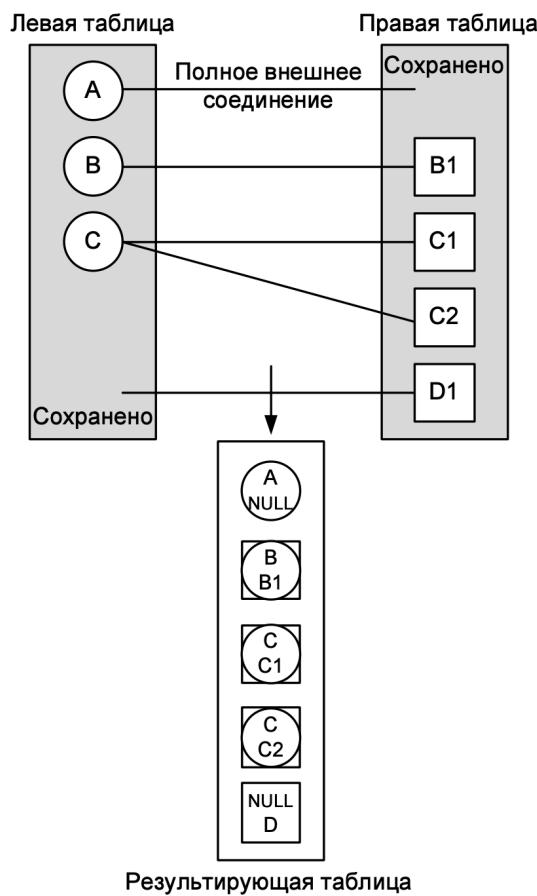


Рис. 4.5: Полное внешнее соединение

КОНТРОЛЬНЫЕ ВОПРОСЫ

1. В чем заключается разница между старым и новым синтаксисом перекрестных соединений?
2. Назовите разные типы внешних соединений.

Ответы на контрольные вопросы

1. В новый синтаксис входят ключевые слова `CROSS JOIN` между именами таблиц, тогда как старый синтаксис использует для этого запятую.
2. Левое, правое и полное соединения.

Резюме занятия

- Перекрестные соединения возвращают декартово произведение строк обеих сторон.
- Внутренние соединения сопоставляют строки с помощью предиката и возвращают только совпадения.

- Внешние соединения сопоставляют строки с помощью предиката и возвращают как совпадения, так и несовпадения из таблиц, помеченных как сохраненные.
- Запросы с мультисоединениями содержат несколько соединений. В них возможно присутствие разных типов соединений. Логический порядок обработки соединений можно контролировать с помощью круглых скобок или изменением положения предложения ON.

Закрепление материала

1. В чем заключается разница между предложениями ON и WHERE?
 - Предложение ON использует двоичную логику, а предложение WHERE — троичную.
 - Предложение ON использует троичную логику, а предложение WHERE — двоичную.
 - Во внешних соединениях предложение ON определяет фильтрацию, а предложение WHERE — сопоставление данных.
 - Во внешних соединениях предложение ON определяет сопоставление данных, а предложение WHERE — фильтрацию.
2. Какие ключевые слова можно опустить в новом стандартном синтаксисе соединений без изменения значения соединения? (Выберите все подходящие варианты.)
 - JOIN.
 - CROSS.
 - INNER.
 - OUTER.

3. Какой синтаксис рекомендуется использовать для перекрестных и внутренних соединений и почему?
 - Синтаксис с ключевым словом JOIN, поскольку это соответствует синтаксису внешнего запроса и вызывает меньше ошибок.
 - Синтаксис с запятой между именами таблиц, поскольку это соответствует синтаксису внешнего запроса и вызывает меньше ошибок.
 - Рекомендуется избегать использования перекрестных и внутренних соединений.
 - Рекомендуется использовать только символы нижнего регистра и опускать ключевые слова по умолчанию, как в случае JOIN вместо INNER JOIN, поскольку они увеличивают потребление энергии.

Ответы

Закрепление материала

1. Правильный ответ: D.
 - A. **Неправильно:** оба предложения используют троичную логику.
 - B. **Неправильно:** оба предложения используют троичную логику.
 - C. **Неправильно:** ON определяет соответствие, а WHERE — фильтрацию.
 - D. **Правильно:** ON определяет соответствие, а WHERE — фильтрацию.

Ответы

653

2. Правильные ответы: С и D.
 - A. **Неправильно:** ключевое слово JOIN не может быть опущено в новом синтаксисе соединений.
 - B. **Неправильно:** если ключевое слово CROSS опущено в CROSS JOIN, одно ключевое слово JOIN означает внутреннее соединение, а не пересекающееся соединение.
 - C. **Правильно:** если ключевое слово INNER опущено в INNER JOIN, значение сохраняется.
 - D. **Правильно:** если ключевое слово OUTER опущено в LEFT OUTER JOIN, RIGHT OUTER JOIN и FULL OUTER JOIN, значение сохраняется.
3. Правильный ответ: A.
 - A. **Правильно:** синтаксис с ключевым словом JOIN согласуется только со стандартным синтаксисом, доступным для внешних соединений, и менее подвержен ошибкам.
 - B. **Неправильно:** внешние соединения не имеют стандартного синтаксиса с использованием запятых.
 - C. **Неправильно:** таких рекомендаций не существует. Пересекающиеся и внутренние соединения имеют право на существование.
 - D. **Неправильно:** таких доказательств нет.

4.2 Использование подзапросов, табличных выражений и оператора APPLY

4.2.1 Независимые подзапросы

Независимые подзапросы — это вложенные запросы, которые не имеют зависимостей от внешнего запроса.

```
1 SELECT productid, productname, unitprice
2 FROM Production.Products
3 WHERE unitprice = (SELECT MIN(unitprice)
4                   FROM Production.Products);
```

Листинг 4.4: Пример вложенного подзапроса

```
1 SELECT productid, productname, unitprice
2 FROM Production.Products
3 WHERE supplierid IN
4   (SELECT supplierid
5    FROM Production.Suppliers
6    WHERE country = N'Japan');
```

Листинг 4.5: Пример вложенного запроса с IN

4.2.2 Коррелированные (связанные) подзапросы

Коррелированные (связанные) подзапросы — это подзапросы, в которых внутренний запрос имеет ссылку на столбец таблицы во внешнем запросе. Их использование несколько сложнее по сравнению с независимыми подзапросами, поскольку невозможно просто выделить внутреннюю часть и запустить ее отдельно.

```
1 SELECT categoryid, productid, productname, unitprice
2 FROM Production.Products AS P1
3 WHERE unitprice = (SELECT MIN(unitprice)
4                   FROM Production.Products AS P2
5                   WHERE P2.categoryid = P1.categoryid);
6
```

```
1 SELECT custid, companyname
2 FROM Sales.Customers AS C
3 WHERE EXISTS
4 (SELECT *
5   FROM Sales.Orders AS O
6   WHERE O.custid = C.custid
7   AND O.orderdate = '20070212');
```

Предикат EXISTS принимает подзапрос на вход и возвращает значение «истина», когда подзапрос возвращает хотя бы одну строку, в противном случае — «ложь».

4.3 Табличные выражения

Табличные выражения — это именованные запросы. Вы пишете внутренний запрос, который возвращает реляционный результирующий набор, даете ему имя и вызываете его из внешнего запроса. Язык T-SQL поддерживает четыре формы табличных выражений:

- производные таблицы;
- обобщенные табличные выражения (common table expression, CTE);
- представления;
- встроенные табличные функции.

ВАЖНО!

Оптимизация табличных выражений

Важно помнить, что с точки зрения производительности, когда SQL Server оптимизирует запросы, содержащие табличные выражения, он сначала извлекает логику табличного выражения и таким образом непосредственно взаимодействует с базовыми таблицами. Он не сохраняет результат табличного выражения во внутренней рабочей таблице и затем не взаимодействует с этой рабочей таблицей. Это означает, что табличные выражения никак не влияют на производительность — ни хорошо, ни плохо, — просто никак.

4.3.1 Производные таблицы

Производная таблица — это, возможно, форма табличного выражения, которая более всего похожа на подзапрос, только это подзапрос, возвращающий в качестве результата целую таблицу.

```
1  SELECT
2      ROW_NUMBER() OVER(PARTITION BY categoryid
3          ORDER BY unitprice, productid) AS rounum,
4      categoryid, productid, productname, unitprice
5  FROM Production.Products;
```

rounum	categoryid	productid	productname	unitprice
1	1	24	Product QOGNU	4.50
2	1	75	Product BWRLG	7.75
3	1	34	Product SWNHY	14.00
4	1	67	Product XLXQF	14.00
5	1	70	Product TOONT	15.00
...				
1	2	3	Product IMEHJ	10.00
2	2	77	Product LUNZZ	13.00
3	2	15	Product KSZOI	15.50
4	2	66	Product LQMGN	17.00
5	2	44	Product VJIEO	19.45
...				

Существует пара проблем, связанных с использованием производных таблиц, которые обусловлены тем, что производная таблица определяется в предложении FROM внешнего запроса. Первая проблема возникает, когда нужно ссылаться на одну производную таблицу из другой. В таком случае приходится прибегать к вложенным производным таблицам, а вложенность часто усложняет логику, затрудняя следование ей и увеличивая возможность появления ошибок.

```
1  SELECT ...
2  FROM (SELECT ...
3    FROM (SELECT ...
4      FROM T1
5      WHERE ...) AS D1
6      WHERE ...) AS D2
7  WHERE ...;
```

Другая проблема производных таблиц имеет отношение к свойству "одновременности" языка. Напомним, что все выражения, появляющиеся в одной и той же фазе логической обработки запроса, концептуально оцениваются в один и тот же момент времени. Это справедливо и для табличных выражений. В результате имя, присвоенное производной таблице, не видно другим элементам, которые появляются в той же самой фазе логической обработки запроса, в которой было определено имя производной таблицы. Это означает, что если вы хотите объединить несколько экземпляров одной производной таблицы, вы не сможете это сделать.

```
1 SELECT ...
2 FROM (SELECT ...
3   FROM T1) AS D1
4   INNER JOIN
5     (SELECT ...
6   FROM T1) AS D2
7   ON ...;
```

4.3.2 Обобщенные табличные выражения

Обобщенное табличное выражение (common table expression, CTE) подобно производной таблице в том смысле, что это именованное табличное выражение, видимое только инструкции, которая его определяет. Так же как запрос к производной таблице, запрос к CTE включает три основных части:

- внутренний запрос;
- имя, которое присваивается запросу и его столбцам;
- внешний запрос.

```
1 WITH <CTE_name>
2 AS ( <inner_query> )
3 <outer_query>;
```

```

1 WITH C AS
2 (SELECT
3     ROW_NUMBER() OVER(PARTITION BY categoryid
4         ORDER BY unitprice, productid) AS rounum,
5     categoryid, productid, productname, unitprice
6     FROM Production.Products)
7 SELECT categoryid, productid, productname, unitprice
8 FROM C
9 WHERE rounum <= 2;

```

Такая структура позволяет составлять более чистый код, который проще понимать. Вам не нужно создавать вложенные СТЕ, как в случае с производными таблицами. Если необходимо определить несколько СТЕ, просто разделите их запятыми.

Отказ от вложенных СТЕ упрощает следование логике запроса и таким образом снижает возможность появления ошибок. Например, если нужно солаться на один СТЕ из другого, можно использовать следующую общую форму:

```

1 WITH C1 AS
2 ( SELECT ...
3   FROM T1
4   WHERE ... ),
5 C2 AS
6 ( SELECT
7   FROM C1
8   WHERE ... )
9 SELECT ...
10  FROM C2
11  WHERE ... ;

```

Поскольку имя СТЕ назначено до начала внешнего запроса, можно солаться на несколько экземпляров того же самого имени СТЕ, что невозможно в случае производных таблиц. Общая форма запроса выглядит следующим образом:

```

1 WITH C AS
2 ( SELECT ...
3   FROM T1 )
4 SELECT ...
5 FROM C AS C1

```

```
6   INNER JOIN C AS C2  
7   ON ...;
```

СТЕ также имеют рекурсивную форму. Тело рекурсивного запроса содержит два или более запросов, обычно разделенных оператором UNION ALL. По крайней мере, один запрос в теле СТЕ, известный как закрепленный элемент, — это запрос, который возвращает реляционный результат. Закрепленный запрос вызывается только один раз. Кроме того, хотя бы один запрос в теле СТЕ, называемый рекурсивным элементом, имеет ссылку на имя СТЕ. Этот запрос вызывается неоднократно, до тех пор, пока он не возвратит пустой результирующий набор.

```
1  WITH EmpsCTE AS  
2  ( SELECT empid, mgrid, firstname, lastname, 0 AS distance  
3    FROM HR.Employees  
4   WHERE empid = 9  
5   UNION ALL  
6   SELECT M.empid, M.mgrid, M.firstname, M.lastname,  
7         S.distance + 1 AS distance  
8    FROM EmpsCTE AS S  
9   JOIN HR.Employees AS M  
10  ON S.mgrid = M.empid )  
11  SELECT empid, mgrid, firstname, lastname, distance  
12  FROM EmpsCTE;
```

4.3.3 Представления и встроенные табличные функции

Чтобы иметь возможность повторного использования, необходимо сохранить определение табличного выражения как объект в базе данных, а для этого можно использовать либо представления, либо встроенные функции, возвращающие табличное значение (табличные функции). Главное различие между представлениями истроенными табличными функциями заключается в том, что первые не принимают входные параметры, а вторые —принимают.

```

1 IF OBJECT_ID('Sales.RankedProducts', 'V') IS NOT NULL
2 DROP VIEW Sales.RankedProducts;
3 GO
4 CREATE VIEW Sales.RankedProducts
5 AS
6 SELECT ROW_NUMBER() OVER(PARTITION BY categoryid
7 ORDER BY unitprice, productid) AS rownum,
8 categoryid, productid, productname, unitprice
9 FROM Production.Products;
10 GO

```

Предположим, вы хотите инкапсулировать логику этого запроса в табличное выражение для повторного использования, а также параметризовать входные данные сотрудника вместо использования константы 9. Этого можно достичнуть с помощью встроенной табличной функции со следующим определением:

```

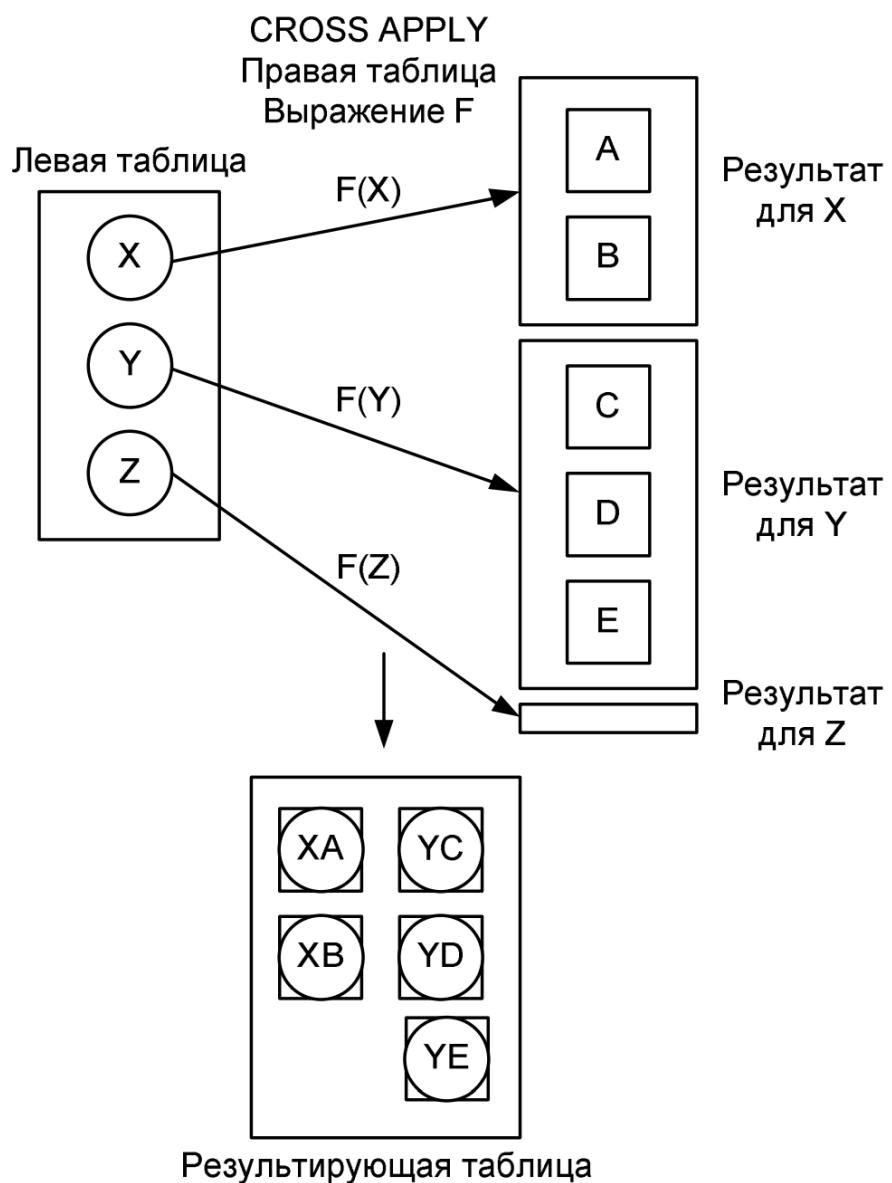
1 IF OBJECT_ID('HR.GetManagers', 'IF') IS NOT NULL DROP FUNCTION HR.
    GetManagers;
2 GO
3 CREATE FUNCTION HR.GetManagers(@empid AS INT) RETURNS TABLE
4 AS
5 RETURN
6 WITH EmpsCTE AS
7 ( SELECT empid, mgrid, firstname, lastname, 0 AS distance
8   FROM HR.Employees
9  WHERE empid = @empid
10 UNION ALL
11   SELECT M.empid, M.mgrid, M.firstname, M.lastname,
12         S.distance + 1 AS distance
13   FROM EmpsCTE AS S
14  JOIN HR.Employees AS M
15    ON S.mgrid = M.empid )
16
17   SELECT empid, mgrid, firstname, lastname, distance
18   FROM EmpsCTE;
19 GO

```

4.4 Оператор APPLY

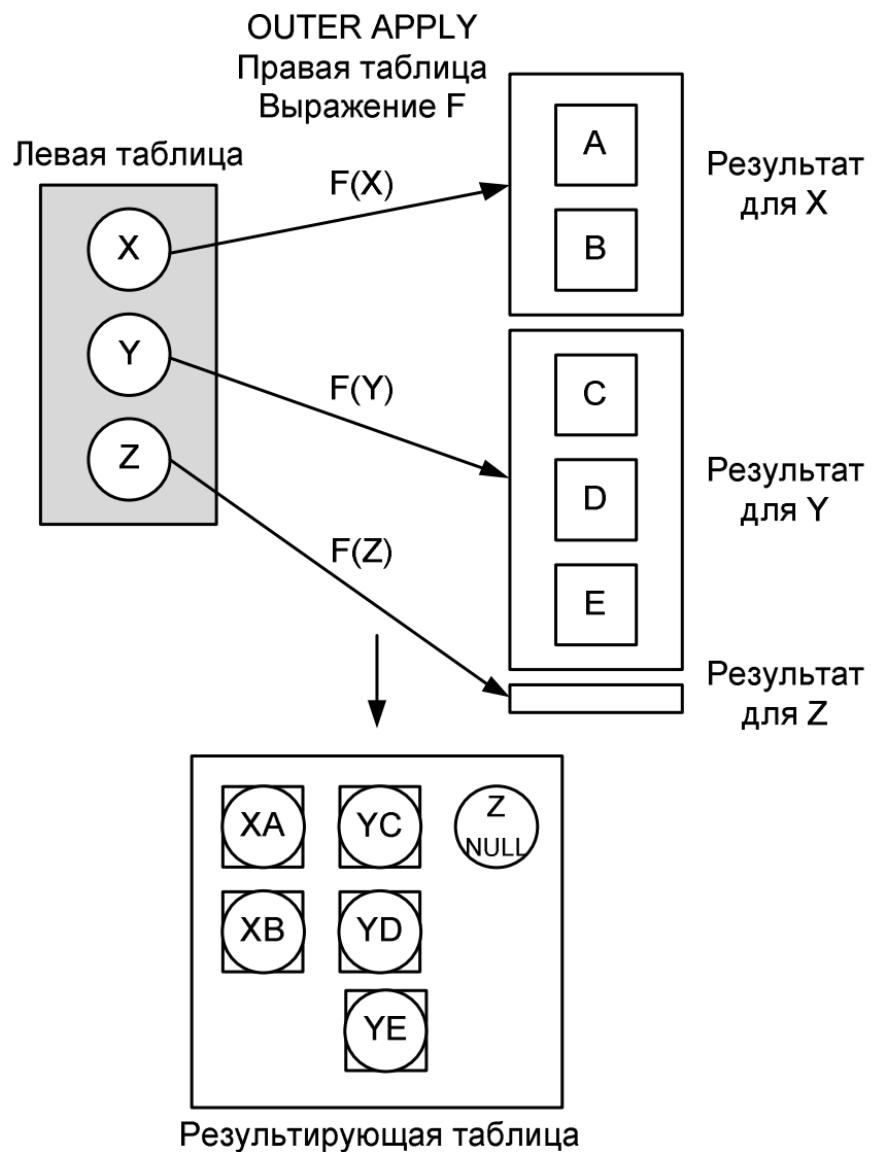
По сравнению с объединением, оператор APPLY интересен тем, что правое табличное выражение может быть связано с левой таблицей; другими словами, внутренний запрос в правом табличном выражении может иметь ссылку на элемент из левой таблицы.

4.4.1 CROSS APPLY



4.4.2 OUTER APPLY

Оператор OUTER APPLY делает то же самое, что и оператор CROSS APPLY, и кроме этого включает в результат строки с левой стороны, которые возвращают пустой набор с правой стороны. Значения NULL используются как заменители для результирующих столбцов с правой стороны. Иными словами, оператор OUTER APPLY сохраняет записи левой стороны.



КОНТРОЛЬНЫЕ ВОПРОСЫ

1. В чем заключается разница между независимым и коррелированным (связанным) подзапросами?
2. В чем заключается разница между операторами **APPLY** и **JOIN**?

Ответы на контрольные вопросы

1. Независимые подзапросы не зависят от внешнего запроса, тогда как связанные подзапросы имеют ссылку на элемент из таблицы во внешнем запросе.
2. При использовании оператора **JOIN** оба входа представляют статические отношения. В операторе **APPLY** левая сторона представляет собой статическое отношение, а правая часть может быть табличным выражением со связями с элементами из левой таблицы.

Практикум

ПРАКТИКУМ Использование подзапросов, табличных выражений и оператора **APPLY**

В данном практикуме вам предстоит применить знания о подзапросах, табличных выражениях и операторе **APPLY**.

Задание 1. Формирование списка продуктов с минимальной ценой за единицу в пределах категории

В этом задании необходимо написать решение, использующее СТЕ для возвращения списка продуктов с наименьшей ценой в пределах категории товара.

1. Откройте SQL Server Management Studio (SSMS) и подключитесь к учебной базе данных TSQL2012.
2. В качестве первого шага напишите запрос к таблице `Production.Products`, который группирует продукты по идентификатору категории `categoryid` и возвращает минимальное значение цены продукта `unitprice` для каждой категории. Следующий запрос реализует данный шаг:

```
SELECT categoryid, MIN(unitprice) AS mn
FROM Production.Products
GROUP BY categoryid;
```

Вы получите такой результат:

categoryid	mn
1	4.50
2	10.00
3	9.20
4	2.50
5	7.00
6	7.45
7	10.00
8	6.00

3. Следующий шаг вашего решения — определить СТЕ на основе предыдущего запроса и затем объединить СТЕ с таблицей Production.Products для того, чтобы возвратить для каждой категории продукты с минимальной ценой за единицу. Этот шаг можно реализовать с помощью следующего кода:

```
WITH CatMin AS
( SELECT categoryid, MIN(unitprice) AS mn
  FROM Production.Products
 GROUP BY categoryid )
SELECT P.categoryid, P.productid, P.productname, P.unitprice
  FROM Production.Products AS P
 INNER JOIN CatMin AS M
    ON P.categoryid = M.categoryid
   AND P.unitprice = M.mn;
```

Этот код представляет законченное решение, возвращающее желаемый результат.

categoryid	productid	productname	unitprice
1	24	Product QOGNU	4.50
2	3	Product IMEHJ	10.00
3	19	Product XKXDO	9.20
4	33	Product ASTMN	2.50
5	52	Product QSRXF	7.00
6	54	Product QAQRL	7.45
7	74	Product BKAZJ	10.00
8	13	Product POXFU	6.00

Задание 2. Формирование списка из N продуктов с минимальными ценами за единицу для каждого поставщика

В этом задании вам предстоит использовать операторы CROSS APPLY и OUTER APPLY.

1. Определите встроенную табличную функцию, которая принимает на вход ID поставщика (@supplierid), а также количество продуктов (@n), и возвращает @n продуктов с минимальными ценами для входного поставщика. При наличии одинаковых элементов в цене продукта используйте ID продукта в качестве уточняющего параметра.

Для определения функции используйте следующий код:

```
IF OBJECT_ID('Production.GetTopProducts', 'IF') IS NOT NULL
DROP FUNCTION
Production.GetTopProducts;
GO
CREATE FUNCTION Production.GetTopProducts(@supplierid AS INT, @n AS BIGINT)
RETURNS TABLE
AS
```

```

RETURN
    SELECT productid, productname, unitprice
    FROM Production.Products
    WHERE supplierid = @supplierid
    ORDER BY unitprice, productid
    OFFSET 0 ROWS FETCH FIRST @n ROWS ONLY;
GO

```

2. Запросите эту функцию, чтобы протестировать ее, указывая ID поставщика равным 1 и число 2 для возвращения двух продуктов с минимальными ценами за единицу для входного поставщика.

```
SELECT * FROM Production.GetTopProducts(1, 2) AS P;
```

Этот код генерирует следующий результат:

productid	productname	unitprice
3	Product IMEHJ	10.00
1	Product HHYDP	18.00

3. Далее, возвратите для каждого поставщика из Японии два продукта с минимальными ценами за единицу. Для этого используйте оператор CROSS APPLY с Production.Suppliers в качестве источника данных с левой стороны и функцией Production.GetTopProducts в качестве правой стороны, как показано далее в примере.

```

SELECT S.supplierid, S.companyname AS supplier, A.*
FROM Production.Suppliers AS S
    CROSS APPLY Production.GetTopProducts(S.supplierid, 2) AS A
WHERE S.country = N'Japan';

```

Этот код генерирует следующий результат:

supplierid	supplier	productid	productname	unitprice
4	Supplier QOVFD	74	Product BKAZJ	10.00
4	Supplier QOVFD	10	Product YHXGE	31.00
6	Supplier QWUSF	13	Product POXFU	6.00
6	Supplier QWUSF	15	Product KSZOI	15.50

4. В предыдущем шаге вы использовали оператор CROSS APPLY, и потому поставщики из Японии, не имеющие связанных с ними продуктов, были отброшены. Пусть вам нужно возвратить и их тоже. Тогда вы должны сохранить левую сторону, и чтобы добиться этого, используйте оператор OUTER APPLY, как показано в примере далее.

```

SELECT S.supplierid, S.companyname AS supplier, A.*
FROM Production.Suppliers AS S
    OUTER APPLY Production.GetTopProducts(S.supplierid, 2) AS A
WHERE S.country = N'Japan';

```

На этот раз выходной набор содержит и поставщиков без продуктов.

supplierid	supplier	productid	productname	unitprice
4	Supplier QOVFD	74	Product BKAZJ	10.00
4	Supplier QOVFD	10	Product YHXGE	31.00
6	Supplier QWUSF	13	Product POXFU	6.00
6	Supplier QWUSF	15	Product KSZOI	15.50
30	Supplier XYZ	NULL	NULL	NULL

5. После этого запустите следующий код для очистки данных:

```

IF OBJECT_ID('Production.GetTopProducts', 'IF') IS NOT NULL
    DROP FUNCTION Production.GetTopProducts;

```

Резюме занятия

- С помощью подзапросов можно вкладывать запросы друг в друга. Можно использовать как независимые, так и связанные подзапросы. Подзапросы могут возвращать в качестве результата одно значение, несколько значений или таблицу.
- Язык T-SQL поддерживает четыре вида табличных выражений, которые являются именованными выражениями запросов. Производные таблицы и CTE — это виды табличных выражений, доступных только для инструкции, в которой они определены. Представления и встроенные табличные функции представляют собой повторно используемые табличные выражения, определения которых сохраняются в базе данных в виде объектов. Представления не поддерживают входные параметры, тогда как встроенные табличные функции поддерживают их.
- Оператор APPLY обрабатывает два табличных выражения как входные наборы. Он применяет правое табличное выражение к каждой строке из левой стороны. Внутренний запрос в правом табличном выражении может иметь связи с элементами из левой таблицы. Оператор APPLY имеет два варианта. Вариант CROSS APPLY не возвращает левые строки, которые возвращают пустой набор из правой стороны. Оператор OUTER APPLY сохраняет левую сторону и поэтому возвращает строки из набора данных с левой стороны, когда правая сторона возвращает пустой набор. Значения NULL используются для замещения атрибутов правой стороны во внешних строках.

Закрепление материала

1. Что происходит, когда скалярный подзапрос возвращает более одного значения?
 - A. Запрос дает ошибку при выполнении.
 - B. Возвращается первое значение.

- C. Возвращается последнее значение.
- D. Результат конвертируется в NULL.
2. В чем преимущество использования СТЕ над производными таблицами? (Выберите все подходящие варианты.)
 - A. СТЕ лучше работают, чем производные таблицы.
 - B. СТЕ не имеют вложенности; код более модульный, что облегчает следование логике кода.
 - C. В отличие от производных таблиц, можно ссылаться на несколько экземпляров одного и того же имени СТЕ, избегая повторения кода.
 - D. В отличие от производных таблиц, СТЕ могут быть использованы всеми инструкциями в сессии, а не только инструкцией, в которых они определены.
3. В чем заключается разница между результатами выражения `T1 CROSS APPLY T2` и `T1 CROSS JOIN T2` (правое табличное выражение не имеет связей с левым)?
 - A. Оператор `CROSS APPLY` фильтрует только строки, в которых значения столбцов с одинаковым именем равны; оператор `CROSS JOIN` просто возвращает все комбинации.
 - B. Если `T1` содержит строки, а `T2` — нет, оператор `CROSS APPLY` возвращает пустой набор, а оператор `CROSS JOIN` все равно возвратит строки из `T1`.
 - C. Если `T1` содержит строки, а `T2` — нет, оператор `CROSS APPLY` все равно возвратит строки из `T1`, а оператор `CROSS JOIN` возвратит пустой набор.
 - D. Между ними нет разницы.

Ответы

Закрепление материала

1. Правильный ответ: А.
 - A. **Правильно:** запрос завершается ошибкой при выполнении, указывая, что возвращено более одного значения.
 - B. **Неправильно:** запрос завершается ошибкой.
 - C. **Неправильно:** запрос завершается ошибкой.
 - D. **Неправильно:** скалярный подзапрос конвертируется в значение NULL, когда он возвращает пустой набор — не несколько значений.
2. Правильные ответы: В и С.
 - A. **Неправильно:** все типы табличных выражений с точки зрения оптимизации обрабатываются одинаково — у них нет вложенности.
 - B. **Правильно:** если нужно сослаться на одну производную таблицу из другой, следует вложить их одна в другую. При использовании СТЕ вы их отделяете друг от друга запятыми, так что код становится модульным и проще для выполнения.
 - C. **Правильно:** поскольку имя СТЕ определено раньше внешнего запроса, который его использует, внешний запрос имеет право ссылаться на несколько экземпляров одного и того же имени СТЕ.
 - D. **Неправильно:** СТЕ видны только той инструкции, которая их определяет.

654

Ответы

3. Правильный ответ: D.
 - A. **Неправильно:** оба возвращают все комбинации.
 - B. **Неправильно:** оба возвращают пустой набор.
 - C. **Неправильно:** оба возвращают пустой набор.
 - D. **Правильно:** оба возвращают одинаковый результат, когда не существует связи, поскольку оператор CROSS APPLY применяет все строки из T2 к каждой строке из T1.

4.5 Использование операторов работы с наборами

Язык T-SQL поддерживает три оператора работы с наборами: UNION, INTERSECT и EXCEPT; также в нем поддерживается оператор UNION ALL для работы с наборами типа multiset.

```
1 <query 1>
2 <set operator>
3 <query 2>
4 [ORDER BY <order_by_list>]
```

Существует несколько правил, которым необходимо следовать при использовании операторов работы с наборами:

- Поскольку полные строки сопоставляются между результирующими наборами, количество столбцов в запросах должно совпадать и типы соответствующих столбцов должны быть сравнимыми (неявно конвертируемыми).
- Операторы работы с наборами рассматривают два значения NULL как равные при сравнении. Это довольно необычно по сравнению с предложениями фильтрации, такими как WHERE и ON.
- Поскольку речь идет об операторах работы с наборами, а не операторах курсора, отдельные запросы не могут иметь предложений ORDER BY.
- При желании можно добавить предложение ORDER BY, которое определяет порядок сортировки результата, возвращаемого оператором работы с наборами.
- Имена столбцов для результирующих столбцов определяются первым запросом.

4.5.1 Операторы UNION и UNION ALL

Оператор работы с наборами UNION объединяет результаты двух входных запросов. В качестве примера использования оператора UNION рассмотрим следующий запрос, который возвращает местоположения сотрудников, клиентов или и тех, и других.

```
1 SELECT country, region, city
2 FROM HR.Employees
3 UNION
```

```
4 | SELECT country, region, city  
5 | FROM Sales.Customers;
```

country	region	city
UK	NULL	London
USA	WA	Kirkland
USA	WA	Seattle
...		
(71 row(s) affected)		

Если вы хотите сохранить дубликаты, например, чтобы позже сгруппировать строки и подсчитать количество вхождений, то вместо оператора UNION следует использовать оператор UNION ALL. Оператор UNION ALL объединяет результаты двух входных запросов, но не пытается удалить дубликаты.



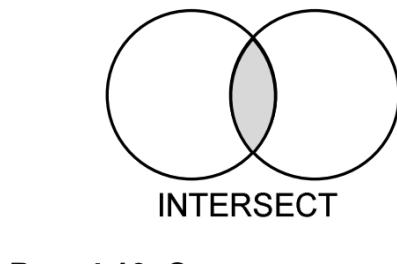
ВАЖНО!

Сравнение операторов UNION и UNION ALL

Если наборы, которые вы объединяете, не пересекаются и появление дубликатов не ожидается, операторы UNION и UNION ALL возвратят одинаковый результат. Но с точки зрения производительности в подобном случае лучше применять оператор UNION ALL, поскольку при использовании оператора UNION SQL Server может попытаться удалить дубликаты, что приведет к ненужным затратам.

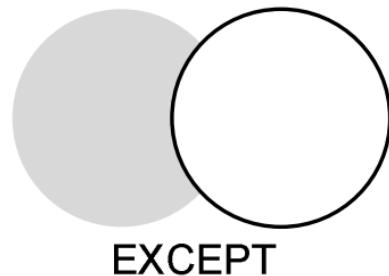
4.5.2 Оператор INTERSECT

Оператор INTERSECT возвращает только различные строки, общие для обоих наборов. Другими словами, если строка хотя бы один раз появляется в первом наборе и хотя бы один раз — во втором, она появится один раз в результате оператора INTERSECT.



4.5.3 Оператор EXCEPT

Оператор EXCEPT позволяет получить разность наборов данных. Он возвращает отличающиеся строки, которые появляются в первом запросе и не появляются во втором. Иными словами, если строка хотя бы раз появляется в первом запросе и ни разу во втором, она один раз возвращается в выходном наборе.



Следующий запрос, который можно использовать в качестве примера использования оператора EXCEPT, возвращает местоположения сотрудников, не являющиеся местоположениями клиентов

```
1 SELECT country, region, city
2 FROM HR.Employees
3 EXCEPT
4 SELECT country, region, city
5 FROM Sales.Customers;
```

КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Какие операторы работы с наборами поддерживает язык T-SQL?
2. Назовите два требования к запросам, участвующим в операторе работы с наборами.

Ответы на контрольные вопросы

1. Операторы работы с наборами UNION, INTERSECT и EXCEPT, а также оператор UNION ALL работают с наборами типа multiset.
2. Количество столбцов в двух запросах должно быть одинаковым, и соответствующие столбцы должны иметь совместимые типы данных.

Резюме занятия

- Операторы работы с наборами сравнивают полные строки в результирующих наборах двух запросов.
- Оператор UNION объединяет входные наборы, возвращая несовпадающие строки.
- Оператор UNION ALL объединяет входные наборы, не удаляя дубликаты.
- Оператор UNION ALL объединяет входные наборы, не удаляя дубликаты.
- Оператор EXCEPT возвращает строки, которые появляются в первом наборе, но не во втором, возвращая несовпадающие строки.

Закрепление материала

1. Какой из следующих операторов удаляет дубликаты из результата? (Выберите все подходящие варианты.)

146

Глава 4

- A. UNION.
 - B. UNION ALL.
 - C. INTERSECT.
 - D. EXCEPT.
2. В каком операторе порядок сортировки во входном запросе имеет значение?
- A. UNION.
 - B. UNION ALL.
 - C. INTERSECT.
 - D. EXCEPT.
3. Какое из приведенных далее выражений является эквивалентом выражения $<query_1> \text{ UNION } <query_2> \text{ INTERSECT } <query_3> \text{ EXCEPT } <query_4>$?
- A. $(<query_1> \text{ UNION } <query_2>) \text{ INTERSECT } (<query_3> \text{ EXCEPT } <query_4>)$.
 - B. $<query_1> \text{ UNION } (<query_2> \text{ INTERSECT } <query_3>) \text{ EXCEPT } <query_4>$.
 - C. $<query_1> \text{ UNION } <query_2> \text{ INTERSECT } (<query_3> \text{ EXCEPT } <query_4>)$.
 - D. $<query_1> \text{ UNION } (<query_2> \text{ INTERSECT } <query_3> \text{ EXCEPT } <query_4>)$.

Ответы

Занятие 3

Закрепление материала

1. Правильные ответы: А, С и D.
 - A. **Правильно:** оператор UNION удаляет дубликаты.
 - B. **Неверно:** оператор UNION ALL не удаляет дубликаты.
 - C. **Правильно:** оператор INTERSECT удаляет дубликаты.
 - D. **Правильно:** оператор EXCEPT удаляет дубликаты.
2. Правильный ответ: D.
 - A. **Неверно:** при использовании оператора UNION порядок входных запросов не имеет значения.
 - B. **Неверно:** при использовании оператора UNION ALL порядок входных запросов не имеет значения.
 - C. **Неверно:** при использовании оператора INTERSECT порядок входных запросов не имеет значения.
 - D. **Правильно:** при использовании оператора EXCEPT порядок входных запросов имеет значение.
3. Правильный ответ: В.
 - A. **Неверно:** без круглых скобок оператор INTERSECT имеет более высокий приоритет над другими операторами; если используются круглые скобки, он оценивается последним.
 - B. **Правильно:** без круглых скобок, оператор INTERSECT имеет более высокий приоритет над другими операторами; то же самое и при использовании круглых скобок.
 - C. **Неверно:** без круглых скобок оператор INTERSECT имеет более высокий приоритет над другими операторами; если используются круглые скобки, первым оценивается оператор EXCEPT.
 - D. **Неверно:** без круглых скобок оператор UNION оценивается вторым (после оператора INTERSECT), а с круглыми скобками оператор UNION оценивается последним.

Упражнения

Упражнение 1. Анализ кода

Вас попросили проанализировать код в системе, в которой существуют проблемы сопровождения кода, а также производительности. Вы обнаружили следующие результаты и должны определить, что рекомендовать клиенту.

1. Вы находите множество запросов, которые используют несколько уровней вложенности производных таблиц, затрудняя следование логике запросов. Вы также обнаруживаете множество запросов, которые объединяют несколько производных таблиц, основанных на одном и том же запросе, а также выясняете, что некоторые запросы повторяются в нескольких местах в коде. Что вы можете порекомендовать клиенту для уменьшения сложности и повышения сопровождения кода?
2. В процессе анализа кода вы устанавливаете множество случаев, в которых используются курсоры для поочередного доступа к экземплярам определенного элемента (такого как клиент, сотрудник, грузоотправитель); далее код вызывает запрос для каждого из этих элементов, сохраняя результат во временной таблице; затем код просто возвращает все строки из этой временной таблицы. Клиент

испытывает проблемы и с сопровождением, и с производительностью существующего кода. Что вы можете порекомендовать?

3. Вы установили проблемы производительности, связанные с соединениями. Вы выяснили, что в системе нет индексов, созданных явно; существуют только индексы, созданные по умолчанию с помощью первичного ключа и ограничений уникальности. Что вы можете порекомендовать?

Упражнение 2. Объяснение операторов работы с наборами

Вы читаете лекцию об операторах работы с наборами на конференции. В конце занятия вы предоставляете аудитории возможность задать вопросы. Ответьте на следующие вопросы, заданные вам участниками лекции.

1. У нас в системе есть множество представлений, использующих оператор `UNION` для комбинирования непересекающихся множеств из разных таблиц. При вызове этих представлений заметно снижение производительности. Что вы посоветуете для улучшения производительности?
2. Укажите, пожалуйста, преимущества использования операторов работы с наборами, таких как `INTERSECT` и `EXCEPT`, по сравнению с применением внутренних и внешних соединений?

Ответы

Упражнения

Упражнение 1. Анализ кода

- Чтобы избавиться от сложности вложенных производных таблиц, в дополнение к дублированию кода производных таблиц, вы можете использовать СТЕ. СТЕ не имеют вложенности; напротив, они обладают большей модульностью. Вы также можете определить СТЕ один раз и ссылаться на него много раз во внешнем запросе. Что касается запросов, которые повторяются в разных частях в коде, для многократного использования кода вы можете применять представления и встроенные табличные функции. Если вам не нужно передавать параметры, используйте первый, если нужно — второй.
- Клиент должен рассмотреть использование оператора `APPLY` вместо курсора и запроса на каждую строку. Оператор `APPLY` требует меньше кода и поэтому улучшает сопровождение, а также он не влечет снижения производительности, которое, как правило, характерно для курсоров.
- Клиент должен проверить связи по внешнему ключу и рассмотреть создание индексов на столбцах внешнего ключа.

Упражнение 2. Объяснение операторов работы с наборами

- Оператор `UNION` возвращает различающиеся строки. Когда объединенные наборы не пересекаются, нет дубликатов, которые надо удалять, но оптимизатор запросов SQL Server может не знать этого. Попытки удалять дубликаты, даже если их нет, приводят к дополнительным затратам. Поэтому когда наборы не пересекаются, важно использовать оператор `UNION ALL`, а не оператор `UNION`. Кроме того, добавление ограничений `CHECK`, которые определяют диапазоны, поддерживаемые каждой таблицей, может помочь оптимизатору понять, что наборы не пересекаются. Тогда даже при использовании оператора `UNION` оптимизатор может понять, что не нужно удалять дубликаты.
- Операторы работы с наборами имеют множество преимуществ. Они позволяют писать более простой код, поскольку не нужно явно сравнивать столбцы двух входных наборов, как при использовании соединений. Также когда операторы работы с наборами сравнивают два значения `NULL`, они считают их равными, что не имеет места в случае соединений. Когда требуется именно такое поведение, проще применять операторы работы с наборами. При использовании соединений для получения такого же поведения надо добавлять предикаты.

5 Группирование и оконные функции

5.1 Написание запросов для группировки данных

```
1  SELECT shipperid,
2    COUNT(*) AS numorders,
3    COUNT(shippeddate) AS shippedorders,
4    MIN(shippeddate) AS firstshipdate,
5    MAX(shippeddate) AS lastshipdate,
6    SUM(val) AS totalvalue
7  FROM Sales.OrderValues
8  GROUP BY shipperid;
```

Обратите внимание на разницу между результатами функций COUNT(shippeddate) и COUNT(*). Первая игнорирует значения NULL в столбце shippeddate, и поэтому дает количество заказов меньшее или равное количеству заказов, полученных во второй функции.

Используя общие функции наборов, можно работать с отдельными вхождениями, указывая предложение DISTINCT перед выражением, как в следующем примере:

```
1  SELECT shipperid, COUNT(DISTINCT shippeddate) AS numshippingdates
2  FROM Sales.Orders
3  GROUP BY shipperid;
```

Опция DISTINCT доступна не только в функции COUNT, но также в других общих функциях наборов. Однако, как правило, она используется с функцией COUNT.

5.2 Работа с несколькими наборами группирования

Язык T-SQL поддерживает три выражения, которые позволяют задавать несколько наборов группирования: GROUPING SETS, CUBE и ROLLUP. Они

используются в предложении GROUP BY.

5.2.1 GROUPING SETS

Выражение GROUPING SETS можно использовать для создания списка наборов группирования, которые требуется определить в запросе. Следующий запрос определяет четыре набора группирования:

```
1 SELECT shipperid, YEAR(shippeddate) AS shipyear, COUNT(*) AS numorders
2 FROM Sales.Orders
3 WHERE shippeddate IS NOT NULL
4 GROUP BY GROUPING SETS
5 (
6 ( shipperid, YEAR(shippeddate)),
7 ( shipperid ),
8 ( YEAR(shippeddate) ),
9 ( )
10 );
```

shipperid	shipyear	numorders
1	2006	36
2	2006	56
3	2006	51
NULL	2006	143
1	2007	130
2	2007	143
3	2007	125
NULL	2007	398
1	2008	79
2	2008	116
3	2008	73
NULL	2008	268
NULL	NULL	809
3	NULL	249
1	NULL	245
2	NULL	315

5.2.2 CUBE

Выражение CUBE принимает список выражений на вход и определяет все возможные наборы группирования, которые могут быть сгенерированы из входов, включая пустой набор группирования. Например, следующий запрос является логическим эквивалентом предыдущего запроса, использовавшего выражение GROUPING SETS.

```
1 SELECT shipperid, YEAR(shippeddate) AS shipyear, COUNT(*) AS numorders
2 FROM Sales.Orders
3 GROUP BY CUBE(shipperid, YEAR(shippeddate));
```

Выражение CUBE определяет все четыре возможных набора группирования из двух входов:

1. (shipperid, YEAR(shippeddate)).
2. (shipperid).
3. (YEAR(shippeddate)).
4. ().

5.2.3 ROLLUP

Выражение ROLLUP также представляет собой сокращенный вариант выражения GROUPING SETS, но оно используется, когда существует иерархическая структура, сформированная входными элементами. В таком случае только поднабор возможных наборов группирования представляет реальный интерес. Рассмотрим, например, иерархию местоположений, созданную в этом заказе элементами shipcountry, shipregion и shipcity. Имеет смысл свертывать данные только в одном направлении, производя статистические вычисления для следующих наборов группирования:

1. (shipcountry, shipregion, shipcity).
2. (shipcountry, shipregion).

3. (shipcountry).

4. ().

Другие наборы группирования просто не интересны. Например, хотя одно и то же название города может появиться в разных местах в мире, нет смысла агрегировать все вхождения этого названия — независимо от региона и страны.

```
1  SELECT shipcountry, shipregion, shipcity, COUNT(*) AS numorders
2  FROM Sales.Orders
3  GROUP BY ROLLUP(shipcountry, shipregion, shipcity);
```

shipcountry	shipregion	shipcity	numorders
Argentina	NULL	Buenos Aires	16
Argentina	NULL	NULL	16
Argentina	NULL	NULL	16
...			
USA	AK	Anchorage	10
USA	AK	NULL	10
USA	CA	San Francisco	4
USA	CA	NULL	4

158

USA	ID	Boise	31
USA	ID	NULL	31
...			
USA	NULL	NULL	122
...			
NULL	NULL	NULL	830

Проблема возникает при определении строк, связанных с одиночным набором группирования, когда сгруппированный столбец разрешает значения NULL — как в случае со столбцом shipregion. Как объяснить, представляет ли

значение NULL в результате замещающее значение (означающее "все регионы") или исходное значение NULL из таблицы (означающее "неприменимый регион")? В языке T-SQL имеются две функции, позволяющие решить эту проблему: GROUPING и GROUPING_ID.

5.2.4 GROUPING, GROUPING_ID

Функция GROUPING принимает единственный элемент на вход и возвращает 0, если этот элемент входит в состав набора группирования, и 1 — когда не входит в него. Следующий запрос демонстрирует использование функции GROUPING:

```
1  SELECT
2    shipcountry, GROUPING(shipcountry) AS grpcountry,
3    shipregion , GROUPING(shipregion) AS grpregion ,
4    shipcity   , GROUPING(shipcity) AS grpcity ,
5    COUNT(*) AS numorders
6  FROM Sales.Orders
7  GROUP BY ROLLUP(shipcountry, shipregion, shipcity);
```

shipcountry	grpcountry	shipregion	grpcountry	shipcity	grpcountry	numorders
Argentina	0	NULL	0	Buenos Aires	0	16
Argentina	0	NULL	0	NULL	1	16
Argentina	0	NULL	1	NULL	1	16
...						
USA	0	AK	0	Anchorage	0	10
USA	0	AK	0	NULL	1	10
USA	0	CA	0	San Francisco	0	4
USA	0	CA	0	NULL	1	4
USA	0	ID	0	Boise	0	31
USA	0	ID	0	NULL	1	31
...						
USA	0	NULL	1	NULL	1	122
...						
NULL	1	NULL	1	NULL	1	830

Другая функция, которую можно использовать для определения наборов группирования — GROUPING_ID. Она принимает на вход список сгруппированных столбцов и возвращает целое число, представляющее битовую карту. Самый правый бит представляет самое правое входное значение. Этот бит

равен 0, когда соответствующий элемент является частью набора группирования, и 1 — в противном случае.

```
1  SELECT GROUPING_ID(shipcountry, shipregion, shipcity) AS grp_id,
2    shipcountry, shipregion, shipcity,
3    COUNT(*) AS numorders
4  FROM Sales.Orders
5  GROUP BY ROLLUP( shipcountry, shipregion, shipcity );
```

grp_id	shipcountry	shipregion	shipcity	numorders
0	Argentina	NULL	Buenos Aires	16
1	Argentina	NULL	NULL	16
3	Argentina	NULL	NULL	16
...				
0	USA	AK	Anchorage	10
1	USA	AK	NULL	10
0	USA	CA	San Francisco	4
1	USA	CA	NULL	4
0	USA	ID	Boise	31
1	USA	ID	NULL	31
...				
3	USA	NULL	NULL	122
...				
7	NULL	NULL	NULL	830

COBET

Подготовка к экзамену

Можно указать несколько выражений GROUPING SETS, CUBE и ROLLUP, разделенных запятыми, в предложении GROUP BY. Этим достигается эффект умножения. Например, выражение CUBE(a, b, c) определяет 8 наборов группирования, а выражение ROLLUP(x, y, z) определяет 4 набора группирования. Разделив их запятой, как в случае CUBE(a, b, c), ROLLUP(x, y, z), вы их перемножаете и получаете в результате 32 набора группирования.

КОНТРОЛЬНЫЕ ВОПРОСЫ

- Что делает запрос группирующим запросом?
- Какие выражения можно использовать для определения нескольких наборов группирования в одном запросе?

Ответы на контрольные вопросы

- Использование статистической функции, предложения GROUP BY или и того, и другого.
- Выражения GROUPING SETS, CUBE и ROLLUP.

Резюме занятия

- Язык T-SQL позволяет группировать данные и выполнять операции анализа данных на группах.
- Можно применять статистические функции, такие как COUNT, SUM, AVG, MIN и MAX, к группам.
- Стандартные групповые запросы определяют только один набор группирования.
- Новую функциональность языка T-SQL можно использовать для определения нескольких наборов группирования в одном запросе с помощью выражений GROUPING SETS, CUBE и ROLLUP.

Закрепление материала

1. Какие ограничения налагаются групповые запросы?
 - A. Если запрос является групповым, обязательно должна вызываться статистическая функция.
 - B. Если в запросе используется статистическая функция, должно присутствовать предложение GROUP BY.

- C. Элементы предложения GROUP BY также должны быть указаны в предложении SELECT.
 - D. При ссылке на элемент из запрашиваемых таблиц в предложениях HAVING, SELECT или ORDER BY, он должен либо появиться в списке GROUP BY, либо содержаться в статистической функции.
2. Каково назначение функций GROUPING и GROUPING_ID? (Выберите все подходящие варианты.)
 - A. Эти функции можно использовать в предложении GROUP BY для группировки данных.
 - B. Эти функции можно использовать, чтобы указать, замещает ли значение NULL в результате запроса элемент, не являющийся частью набора группирования, или это оригинальное значение NULL из таблицы.
 - C. Эти функции можно использовать, чтобы уникальным образом определить набор группирования, с которым связана результирующая строка.
 - D. Эти функции можно использовать для сортировки данных на основании ассоциации набора группирования, т. е. сначала детализация, а затем агрегирование.
 3. В чем заключается разница между статистической функцией COUNT(*) и базовой функцией COUNT(<expression>)?
 - A. Функция COUNT(*) подсчитывает строки; функция COUNT(<expression>) подсчитывает строки, в которых <expression> не равно значению NULL.
 - B. Функция COUNT(*) подсчитывает столбцы; функция COUNT(<expression>) подсчитывает строки.
 - C. Функция COUNT(*) возвращает тип данных BIGINT; функция COUNT(<expression>) возвращает тип данных INT.
 - D. Между этими функциями не существует разницы.

Ответы

Закрепление материала

1. Правильный ответ: D.
 - A. **Неправильно:** можно группировать строки без вызова статистической функции.
 - B. **Неправильно:** запрос может включать статистическую функцию без предложения GROUP BY. Если группирование подразумевается — все строки составляют одну группу.
 - C. **Неправильно:** не существует требования, чтобы элементы группирования появлялись в списке SELECT, хотя, как правило, возвращают элементы, которые группируются.
 - D. **Правильно:** запрос группирования возвращает только одну строку на группу. Поэтому все выражения, появляющиеся на этапах, которые оцениваются после предложения GROUP BY (HAVING, SELECT и ORDER BY), должны гарантировать возвращение одного значения на группу. Отсюда и происходит это ограничение.
2. Правильные ответы: B, C и D.
 - A. **Неправильно:** эти функции не могут быть использованы в предложении GROUP BY.
 - B. **Правильно:** когда функции возвращают 1 бит, для замещения используется значение NULL; когда они возвращают 0 бит, значение NULL происходит из таблицы.
 - C. **Правильно:** каждый набор группирования может быть указан уникальной комбинацией 1 и 0, возвращенной этими функциями.
 - D. **Правильно:** эти функции можно использовать для сортировки, поскольку они возвращают 0 бит для детального элемента и 1 бит для агрегированного элемента. Поэтому если нужно сначала видеть детали, следует сортировать по результату функции в порядке возрастания.
3. Правильный ответ: A.
 - A. **Правильно:** функция COUNT(*) не обрабатывает входное выражение; она вычисляет число строк в группе. Функция COUNT(<expression>) обрабатывает выражение и игнорирует значения NULL. Интересно то, что функция COUNT(<expression>) возвращает 0, когда все входные значения — NULL, тогда как другие общие функции работы с наборами данных, такие как MIN, MAX, SUM, и AVG, возвращают в подобном случае значение NULL.
 - B. **Неправильно:** функция COUNT(*) подсчитывает строки.
 - C. **Неправильно:** функция COUNT(*) подсчитывает строки.
 - D. **Неправильно:** очевидно, что эти функции отличаются в смысле обработки значений NULL.

5.3 Сведение и отмена сведения данных

5.3.1 Сведение данных

Сведение — это технология группирования и агрегирования данных путем трансформации их из состояния строк в состояние столбцов. Во всех запросах сведения необходимо указать три элемента:

- Что вы хотите видеть в строках? Этот элемент называется on rows, или группирующий элемент (grouping element).
- Что вы хотите видеть в столбцах? Этот элемент называется on cols, или распределяющий элемент (spreading element).
- Что вы хотите видеть на пересечении каждого отдельного значения строки и столбца? Этот элемент известен как данные (data), или агрегатный элемент (aggregation element).

```
1 WITH PivotData AS
2 ( SELECT
3   <grouping column>,
4   <spreading column>,
5   <aggregation column>
6   FROM <source table>
7 )
8 SELECT <select list>
9 FROM PivotData
10 PIVOT(<aggregate function>(<aggregation column>)
11 FOR <spreading column> IN (<distinct spreading values>)) AS P;
```

```
1 WITH PivotData AS
2 ( SELECT
3   custid,
4   shipperid,
5   freight
6   FROM Sales.Orders )
7 SELECT custid, [1], [2], [3]
8 FROM PivotData
9 PIVOT(SUM(freight) FOR shipperid IN ([1],[2],[3])) AS P;
```

Листинг 5.1: Вывод суммы заказа у каждого поставщика

5.3.2 Отмена сведения данных

При отмене сведения данных входные данные разворачиваются из состояния столбцов в состояние строки.

Помните, в каждой задаче отмены сведения необходимо указать следующие три элемента:

- набор исходных столбцов, для которых нужно выполнить отмену сведения (в данном случае [1], [2], [3]);
- имя, которое будет присвоено столбцу с полученными значениями (в данном случае freight);
- имя, которое вы хотите присвоить столбцу целевых имен (в данном случае shipperid).

```
1 SELECT <column list>, <names column>, <values column>
2 FROM <source table>
3 UNPIVOT(<values column> FOR <names column> IN(<source columns>)) AS U;
```

```
1 SELECT custid, shipperid, freight
2 FROM Sales.FreightTotals
3 UNPIVOT(freight FOR shipperid IN([1],[2],[3])) AS U;
```

Контрольные вопросы

1. В чем заключается разница между операторами PIVOT и UNPIVOT?
2. В виде каких конструкций реализованы операторы PIVOT и UNPIVOT?

Ответы на контрольные вопросы

1. Оператор PIVOT выполняет ротацию данных из состояния строк в состояние столбцов; оператор UNPIVOT разворачивает данные из столбцов в строки.
2. Операторы PIVOT и UNPIVOT реализуются как табличные операторы.

Резюме занятия

- Сведение данных — это особая форма группирования и агрегирования данных, когда данные разворачиваются из состояния строк в состояние столбцов.
- При сведении данных необходимо указать три вещи: группирующий элемент, распределяющий элемент и агрегатный элемент.
- T-SQL поддерживает собственный оператор PIVOT, используемый для сведения данных входной таблицы.
- Отмена сведения данных трансформирует данные из состояния столбцов в состояние строк.
- Чтобы выполнить отмену сведения данных, необходимо указать три вещи: исходные столбцы, для которых нужно выполнить отмену сведения, столбец целевых имен и столбец целевых значений.
- T-SQL поддерживает собственный оператор с именем UNPIVOT, который применяется для отмены сведения данных входной таблицы.

Закрепление материала

1. Как определяет оператор `PIVOT`, что является группирующим элементом?
 - A. Это элемент, указанный на входе функции `GROUPING`.
 - B. Это определяется методом исключения — элементы из запрашиваемой таблицы, которые не были указаны как распределяющие и агрегатные элементы.
 - C. Это элемент, указанный в предложении `GROUP BY`.
 - D. Это первичный ключ.
2. Что из перечисленного ниже не разрешено в спецификации оператора `PIVOT`? (Выберите все подходящие варианты.)
 - A. Определение вычисления в качестве входа для статистической функции.
 - B. Определение вычисления в качестве распределяющего элемента.
 - C. Использование подзапроса в предложении `IN`.
 - D. Использование нескольких статистических функций.
3. Какой тип данных имеет столбец целевых значений в результате оператора `UNPIVOT`?
 - A. `INT`.
 - B. `NVARCHAR(128)`.
 - C. `SQL_VARIANT`.
 - D. Тип данных исходных столбцов, к которым применяется отмена сведения.

Ответы

1. Правильный ответ: В.

A. **Неправильно:** функция GROUPING имеет отношение к наборам группирования — не к сведению данных.

Ответы

657

B. **Правильно:** оператор PIVOT определяет группирующий элемент методом исключения, как остающийся после выделения распределяемого и агрегатного элементов.

C. **Неправильно:** при использовании оператора PIVOT группирование для сведения данных происходит как часть оператора PIVOT — перед тем, как оценивать предложение GROUP BY.

D. **Неправильно:** оператор PIVOT не рассматривает определения ограничений для определения элемента группирования.

2. Правильные ответы: А, В, С и D.

A. **Правильно:** нельзя задавать вычисление как вход для статистической функции, а только лишь имя столбца из входной таблицы.

B. **Правильно:** нельзя задавать вычисление как распределяемый элемент, а только лишь имя столбца из входной таблицы.

C. **Правильно:** нельзя задавать подзапрос в предложении IN, а только лишь статический список.

D. **Правильно:** нельзя указывать несколько статистических функций, а только одну.

3. Правильный ответ: D.

A. **Неправильно:** тип данных столбца значений не всегда INT.

B. **Неправильно:** тип данных столбца значений NVARCHAR(128) — это относится к столбцу имен.

C. **Неправильно:** тип данных столбца значений — не SQL_VARIANT.

D. **Правильно:** тип данных столбца значений такой же, как тип данных столбцов, к которым применяется отмена сведения, поэтому они должны все иметь одинаковый тип данных.

5.4 Использование оконных функций

5.4.1 Статистические оконные функции

Следующий запрос вычисляет для каждого заказа процент стоимости текущего заказа от суммарной стоимости по клиенту, а также процент от полной стоимости.

```
1  SELECT custid, orderid, val,
2    CAST(100.0 * val / SUM(val) OVER(PARTITION BY custid) AS NUMERIC(5, 2)) AS
3      pctcust,
4    CAST(100.0 * val / SUM(val) OVER() AS NUMERIC(5, 2)) AS pcttotal
5  FROM Sales.OrderValues;
```

Оконные агрегатные функции поддерживают еще одну возможность фильтрации, называемую кадрированием. Смысл ее заключается в том, что определяется упорядочение внутри секции с помощью предложения оконного кадра и затем, на основе этого упорядочения, можно заключить набор строк между двумя границами.

В предложении оконного кадра указываются единицы измерения (ROWS или RANGE) и экстент оконного кадра (смещение границ по отношению к текущей строке). Используя предложение ROWS, можно указать границы как один из следующих параметров:

- UNBOUNDED PRECEDING или FOLLOWING, означающие начало или конец секции соответственно;
- CURRENT ROW, представляющий текущую строку;
- <n> ROWS PRECEDING или FOLLOWING, означающие n строк перед или после текущей строки соответственно.

Пусть вам нужно отфильтровать результат запроса, возвратив только те строки, в которых промежуточная сумма меньше 1000,00. Следующий код решает эту задачу путем определения обобщенного табличного выражения (common table expression, CTE), используя предыдущий запрос и затем выполняя фильтрацию во внешнем запросе.

```

1  WITH RunningTotals AS
2  ( SELECT custid, orderid, orderdate, val,
3    SUM(val) OVER(PARTITION BY custid
4    ORDER BY orderdate, orderid
5    ROWS BETWEEN UNBOUNDED PRECEDING
6    AND CURRENT ROW) AS runningtotal
7    FROM Sales.OrderValues )
8  SELECT *
9  FROM RunningTotals
10 WHERE runningtotal < 1000.00;

```

Еще один пример ограничителя оконного кадра: если было бы нужно, чтобы кадр включал только последние 3 строки, следовало бы использовать формат ROWS BETWEEN 2 PRECEDING AND CURRENT ROW.

Что касается ограничителя оконного кадра RANGE, согласно стандартному языку SQL, он позволяет определять ограничители на основе логических смещений от ключа сортировки текущей строки.

Помните, что предложение ROWS определяет ограничители — число строк от текущей строки, — на основе физических смещений.

ВАЖНО!

Сравнение предложений ROWS и RANGE

В SQL Server 2012 предложение ROWS обычно оптимизируется значительно лучше, чем RANGE при использовании тех же ограничителей. Если окно определено с помощью предложения упорядочивания окна, но без предложения оконного кадра, по умолчанию принимается RANGE BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW. Поэтому, если не требуется специальное поведение, обеспеченное опцией RANGE, которая включает одноранговые строки, убедитесь в том, что вы явно используете опцию ROWS.

5.4.2 Ранжирующие оконные функции

T-SQL поддерживает четыре ранжирующие оконные функции: ROW_NUMBER, RANK, DENSE_RANK и NTILE.

Следующий запрос демонстрирует использование этих функций.

```

1  SELECT custid, orderid, val,
2  ROW_NUMBER() OVER(ORDER BY val) AS rnum,
3  RANK() OVER(ORDER BY val) AS rnk,
4  DENSE_RANK() OVER(ORDER BY val) AS densernk,
5  NTILE(100) OVER(ORDER BY val) AS ntile100

```

```
6 |     FROM Sales.OrderValues;
```

Этот запрос генерирует следующий результат (показанный здесь в сокращенном виде):

custid	orderid	val	rownum	rnk	densernk	ntile100
12	10782	12.50	1	1	1	1
27	10807	18.40	2	2	2	1
66	10586	23.80	3	3	3	1
76	10767	28.00	4	4	4	1
54	10898	30.00	5	5	5	1
88	10900	33.75	6	6	6	1
48	10883	36.00	7	7	7	1
41	11051	36.00	8	7	7	1
71	10815	40.00	9	9	8	1

ВАЖНО!

Упорядочение представления или упорядочение окна

Поскольку рассматриваемый запрос не содержит предложение ORDER BY для представления данных, то нет гарантии, что строки будут представлены в каком-то определенном порядке. Предложение порядка окна определяет только упорядочение для вычисления оконной функции. Если вы вызываете оконную функцию в запросе, но не указываете предложение ORDER BY для представления, невозможно гарантировать, что строки будут представлены в том же порядке, как для оконной функции. Если такая гарантия необходима, следует добавить предложение ORDER BY для представления данных.

Функция ROW_NUMBER вычисляет уникальный последовательный номер строки, начиная с 1, в секции окна на основании сортировки окна. Поскольку рассматриваемый в примере запрос не содержит предложение секционирования окна, эта функция рассматривает результирующий набор как одну секцию; следовательно, функция присваивает уникальные номера строк для всего результирующего набора запроса.

Функция RANK возвращает число строк в секции, которые имеют более низкое значение сортируемого столбца, чем текущее, плюс 1.

Функция DENSE_RANK возвращает количество предшествующих отличающихся значений упорядочиваемого столбца с более низким значением, чем текущее, плюс 1.

Функция NTILE позволяет организовать строки внутри секции в запрашиваемое количество групп одинакового размера на основе указанной сортировки.

СОВЕТ

Подготовка к экзамену

Как говорилось при обсуждении статистических оконных функций, оконные функции разрешены только в предложениях SELECT и ORDER BY запроса. Если нужно на них сослаться в других предложениях, например в предложении WHERE, следует использовать табличное выражение, такое как CTE. Тогда нужно вызвать оконную функцию в предложении SELECT внутреннего запроса, присвоив выражению псевдоним столбца. Затем можно сослаться на этот псевдоним столбца в предложении WHERE внешнего запроса. Вы можете попробовать этот способ в заданиях к этому занятию.

5.4.3 Оконные функции смещения

Оконные функции смещения возвращают элемент строки, которая находится на указанном смещении от текущей строки в секции окна или от первой либо последней строки оконного кадра. T-SQL поддерживает следующие оконные функции смещения: LAG, LEAD, FIRST_VALUE и LAST_VALUE.

Функции LAG и LEAD используют смещение относительно текущей строки, а функции FIRST_VALUE и LAST_VALUE работают с первой или последней строкой окна соответственно.

Функция LAG возвращает элемент строки в текущей секции, который является запрашиваемым числом строк перед данной строкой (на основании сортировки окна), при этом смещение по умолчанию принимается равным 1. Функция LEAD возвращает элемент строки, который находится на запрашиваемом смещении после текущей строки.

```
1  SELECT custid, orderid, orderdate, val,
2    LAG(val) OVER(PARTITION BY custid
3      ORDER BY orderdate, orderid) AS prev_val,
4    LEAD(val) OVER(PARTITION BY custid
5      ORDER BY orderdate, orderid) AS next_val
6  FROM Sales.OrderValues;
```

При желании иметь смещение, отличное от 1, следует указать его вторым аргументом, как в случае LAG(val, 3).

custid	orderid	orderdate	val	prev_val	next_val
1	10643	2007-08-25	814.50	NULL	878.00
1	10692	2007-10-03	878.00	814.50	330.00
1	10702	2007-10-13	330.00	878.00	845.80
1	10835	2008-01-15	845.80	330.00	471.20
1	10952	2008-03-16	471.20	845.80	933.50
1	11011	2008-04-09	933.50	471.20	NULL
2	10308	2006-09-18	88.80	NULL	479.75
2	10625	2007-08-08	479.75	88.80	320.00
2	10759	2007-11-28	320.00	479.75	514.40
2	10926	2008-03-04	514.40	320.00	NULL
...					

Заметьте, что если строка не существует на запрашиваемом смещении, функция возвращает по умолчанию значение NULL. Если в этом случае нужно возвращать другое значение, его следует указать в качестве третьего элемента, как в случае LAG(val, 3, 0).

ВАЖНО!

Кадр по умолчанию и производительность параметра RANGE

Напомним, что когда оконный кадр применим к функции, но явно не указано предложение оконного кадра, по умолчанию используется выражение RANGE BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW. Для улучшения производительности, как правило, рекомендуется избегать опции RANGE; для этого требуется явное задание предложения ROWS. Также, если вы хотите получить первую строку в секции, использование функции FIRST_VALUE с рамкой по умолчанию, по меньшей мере, даст правильный результат. Однако если вы хотите получить последнюю строку в секции, использование функции LAST_VALUE с рамкой по умолчанию не даст желаемого результата, поскольку последняя строка в рамке по умолчанию — это текущая строка. Таким образом, используя функцию LAST_VALUE, нужно явно задавать рамку окна, для того чтобы получить то, что вам нужно. И если вам требуется элемент из последней строки секции, вторым разделителем в рамке должен быть UNBOUNDED FOLLOWING.

КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Какие предложения поддерживаются различными видами оконных функций?
2. Что представляют разделители UNBOUNDED PRECEDING и UNBOUNDED FOLLOWING?

Ответы на контрольные вопросы

1. Предложения секционирования, упорядочения и кадрирования.
2. Начало и конец секции соответственно.

Резюме занятия

- Оконные функции выполняют аналитические вычисления данных. Они обрабатывают набор строк, определенный для каждой базовой строки, с помощью предложения OVER.
- Язык T-SQL поддерживает статистические, ранжирующие оконные функции, а также оконные функции смещения. Все оконные функции поддерживают предложения секционирования и сортировки. Статистические оконные функции, кроме FIRST_VALUE и LAST_VALUE, также поддерживают предложение оконного кадра.
- В отличие от группирующих запросов, которые скрывают строки детализации и возвращают только одну строку на группу, оконные запросы не скрывают эти строки. Они возвращают одну строку на каждую строку в базовом запросе и позволяют смешивать элементы детализации и оконные функции в одних и тех же выражениях.

Закрепление материала

оконные функции, можно найти в приложении "Справка о книге".

1. Какие границы используют по умолчанию оконные функции, когда указано предложение упорядочивания окна, но не указано явное предложение оконного кадра? (Выберите все подходящие варианты.)
 - A. ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW.
 - B. ROWS UNBOUNDED PRECEDING.

¹ Бен-Ган И. Microsoft SQL Server 2012. Высокопроизводительный код T-SQL. Оконные функции: Пер. с англ. — М.: Издательство "Русская редакция"; СПб.: БХВ-Петербург, 2013. — 256 стр. : ил.

- C. RANGE BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW.
 - D. RANGE UNBOUNDED PRECEDING.
2. Что вычисляют функции RANK и DENSE_RANK?
 - A. Функция RANK возвращает количество строк, которые имеют меньшее значение упорядочения (при сортировке по возрастанию), чем текущее; функция DENSE_RANK возвращает число отличающихся значений упорядочения, меньших, чем текущее.
 - B. Функция RANK возвращает число строк, имеющих меньшее значение упорядочения, чем текущее, плюс 1; функция DENSE_RANK возвращает число отличных значений упорядочения, меньших, чем текущее, плюс 1.
 - C. Функция RANK возвращает число строк, имеющих меньшее значение упорядочения, чем текущее, минус 1; функция DENSE_RANK возвращает число отличных значений упорядочения, меньших, чем текущее, минус 1.
 - D. Обе функции возвращают одинаковый результат, если сортировка не является уникальной.
 3. Почему оконные функции разрешены только в предложениях запроса SELECT и ORDER BY?
 - A. Поскольку предполагается, что они воздействуют на результат базового запроса, который получается, когда логическая обработка запроса достигает фазы SELECT.
 - B. Потому что у компании Microsoft не было времени для их реализации в других предложениях.
 - C. Потому что никогда не возникает необходимость фильтрации или группировки данных на основании результата оконных функций.
 - D. Потому что в других предложениях эти функции рассматриваются как функции-лазейки (программы несанкционированного доступа, door-backdoor functions).

Ответы

Справка >

Закрепление материала

1. Правильные ответы: С и D.
 - A. **Неправильно:** ограничение по умолчанию использует единицу измерения окна RANGE.
 - B. **Неправильно:** ограничение по умолчанию использует единицу измерения окна RANGE.
 - C. **Правильно:** это ограничение по умолчанию.
 - D. **Правильно:** это сокращенная форма ограничения по умолчанию, имеющая тот же смысл.
2. Правильный ответ: В.
 - A. **Неправильно:** эти определения на 1 меньше правильных.
 - B. **Правильно:** это правильные определения.

658

Ответы

- C. **Неправильно:** эти определения на 2 меньше правильных.
- D. **Неправильно:** правильным является противоположное утверждение — эти две функции возвращают одинаковый результат при уникальной сортировке.
3. Правильный ответ: А.
 - A. **Правильно:** оконные функции должны обрабатывать результирующий набор базового запроса. С точки зрения логической обработки запросов, этот результирующий набор получается на этапе SELECT.
 - B. **Неправильно:** стандартный язык SQL определяет это ограничение, так что с недостатком времени компании Microsoft это никак не связано.
 - C. **Неправильно:** существуют практические основания для фильтрации или группировки данных с помощью оконных функций.
 - D. **Неправильно:** в SQL нет программ несанкционированного доступа (door-backdoor functions).

Вернуться

Упражнения

Упражнение 1. Усовершенствование операций анализа данных

Вы работаете аналитиком данных в финансовой компании, которая использует SQL Server 2012 для работы с базами данных. Компания недавно обновила СУБД с версии SQL Server 2000. Вы часто используете запросы T-SQL к базе данных компа-

Группирование и оконные функции

187

нии для анализа данных. До сих пор вы были ограничены требованием, чтобы код был совместим с SQL Server 2000, в основном использующий соединения, подзапросы и сгруппированные запросы. Ваши запросы часто были сложными и медленно работали. Сейчас вы оцениваете возможность использования функциональности, доступной в SQL Server 2012.

1. Вам часто приходится вычислять такие вещи, как промежуточные суммы, выполнять расчеты с начала года по текущую дату и скользящее среднее. Что вы теперь можете выбрать для их обработки? Чего нужно остерегаться для обеспечения хорошей производительности?
2. Иногда вам нужно создавать сводные отчеты, где данные преобразуются из строк в столбцы или наоборот. До сих пор вы импортировали данные в Microsoft Excel и решали эти задачи там, но теперь вы предпочитаете делать это в T-SQL. Что вы рассматриваете как средство решения этой задачи? С чем нужно соблюдать осторожность при использовании рассматриваемой вами функциональности?
3. Во многих запросах вам нужно вычислять интервалы времени, т. е. указывать время, прошедшее между предыдущим и текущим событием или между текущим и последующим событием. До сих пор для этого вы использовали вложенные запросы. Что вы рассматриваете сейчас в качестве альтернативы?

Упражнение 2. Собеседование на вакансию разработчика

Вы проходите собеседование на вакансию разработчика T-SQL. Ответьте на следующие вопросы, задаваемые вам проводящим собеседование специалистом.

1. Опишите разницу между функциями `ROW_NUMBER` и `RANK`.
2. Опишите разницу между единицами рамки окна `ROWS` и `RANGE`.
3. Почему вы не можете ссылаться на оконную функцию в предложении `WHERE` запроса и как можно обойти эту ситуацию?

Ответы

Упражнение 1. Усовершенствование операций анализа данных

1. Статистические оконные функции прекрасно подходят для таких вычислений. Что касается вещей, которых следует остерегаться, в текущей реализации SQL Server 2012 следует избегать использования единицы измерения окна RANGE. А также помнить, что без явного предложения оконного кадра вы будете использовать вариант RANGE по умолчанию, поэтому следует явно указывать параметр ROWS.
2. Операторы PIVOT и UNPIVOT подходят для запросов к перекрестным таблицам. При использовании оператора PIVOT следует помнить, что группирующий элемент определяется методом исключения — это то, что осталось от входной таблицы и не было определено ни как распределяющий, ни как агрегатный элемент. Поэтому рекомендуется всегда определять табличное выражение, возвращающее группирующий, распределяющий и агрегатный элементы, и использовать эту таблицу как вход для оператора PIVOT.
3. Функции LAG и LEAD вполне естественны для этой цели.

Упражнение 2. Прохождение собеседования на позицию разработчика

1. Функция ROW_NUMBER не чувствительна к связям в значениях сортировки окна. Поэтому такое вычисление является детерминированным только тогда, когда сортировка окна уникальна. Если сортировка окна неуникальна, эта функция не является детерминированной. Функция RANK чувствительна к связям и дает одинаковое значение ранга для всех строк с одинаковым значением сортировки. Поэтому она является детерминированной, даже если сортировка окна неуникальна.

2. Разница между предложениями ROWS и RANGE в действительности подобна разнице между ROW_NUMBER и RANK соответственно. Если сортировка окна неуникальна, ROWS не включает одноранговые члены и, таким образом, не является детерминированной, тогда как RANGE включает одноранговые члены и, таким образом, является детерминированной. Также опция ROWS может быть оптимизирована с эффективной подкачкой "в памяти"; опция RANGE оптимизируется с подкачкой "на диске" и, следовательно, как правило, является более медленной.
3. Оконные функции разрешены только в предложениях SELECT и ORDER BY, поскольку исходное окно, с которым они должны работать, является результирующим набором базового запроса. Если вам нужно отфильтровать строки с помощью оконной функции, следует использовать табличное выражение, такое как CTE или производная таблица. Нужно указать оконную функцию в предложении SELECT внутреннего запроса и присвоить псевдоним целевому столбцу. Затем можно фильтровать строки, ссылаясь на этот псевдоним столбца в предложении WHERE внешнего запроса.

6 Запросы с полнотекстовым поиском данных

6.1 Создание полнотекстовых каталогов и индексов

Полнотекстовый поиск (full-text search) позволяет выполнять приблизительный поиск в базах данных SQL Server 2012. Прежде чем начать использовать полнотекстовые предикаты и функции, необходимо создать полнотекстовые индексы внутри полнотекстовых каталогов.

6.1.1 Компоненты полнотекстового поиска

Прежде всего, следует проверить, установлен ли компонент FullText Search (Полнотекстовый поиск), выполнив следующий запрос:

```
1 SELECT SERVERPROPERTY('IsFullTextInstalled');
```

Средство разбиения по словам определяет отдельные слова (токены). Токены вставляются в полнотекстовый индекс в сжатом формате. Парадигматический модуль генерирует гибкие формы слова на основании правил данного языка.

Полнотекстовые индексы хранятся в полнотекстовых каталогах. Полнотекстовый каталог — это виртуальный объект, контейнер для полнотекстовых индексов. Как виртуальный объект, он не принадлежит ни к одной файловой группе.

Резюме занятия

- Можно создавать полнотекстовые каталоги с помощью механизмов полнотекстового поиска и семантического поиска SQL Server.

- Можно улучшить полнотекстовый поиск, добавив стоп-слова в стоп-списки, расширив тезаурус разрешив поиск по свойствам документа.
- Можно использовать объект динамического управления sys.dm_fts_parser для того, чтобы проверить, как полнотекстовый поиск разбивает документ на слова, создает словоформы слов и т. д.

Закрепление материала

1. Какие элементы полнотекстового поиска можно использовать для предотвращения индексирования лишних слов (noisy words)? (Выберите все подходящие варианты.)
 - А. Стоп-слова.
 - Б. Тезаурус.
 - С. Парадигматический модуль.
 - Д. Стоп-список.
2. Какую базу вы должны установить для того, чтобы разрешить семантический поиск?
 - А. msdb.
 - Б. distribution.
 - С. semanticsdb.
 - Д. tempdb.
3. Как можно создать синонимы для искомых слов?
 - А. Редактировать файл тезауруса.
 - Б. Создать таблицу тезауруса.
 - С. Использовать стоп-слова и для синонимов.
 - Д. Полнотекстовый поиск не поддерживает синонимы.

Ответы

Закрепление материала

1. Правильные ответы: А и D.
 - A. **Правильно:** стоп-слова содержат лишние слова.
 - B. **Неправильно:** тезаурус используется для синонимов.
 - C. **Неправильно:** парадигматический модуль используется генерации словоформ слов.
 - D. **Правильно:** стоп-слова группируются в стоп-списки.
2. Правильный ответ: С.
 - A. **Неправильно:** база данных msdb устанавливается по умолчанию и используется агентом SQL Server.
 - B. **Неправильно:** база данных распространителя устанавливается по умолчанию и используется для репликации.
 - C. **Правильно:** вам нужна база данных semanticsdb для того, чтобы разрешить семантический поиск.
 - D. **Неправильно:** база данных tempdb устанавливается по умолчанию и используется для всех временных объектов.
3. Правильный ответ: А.
 - A. **Правильно:** можно добавить синонимы, редактируя файл тезауруса.
 - B. **Неправильно:** полнотекстовый поиск применяет для синонимов файлы тезауруса и не использует для этого таблицы.

- C. **Неправильно:** нельзя использовать стоп-слова для синонимов.
- D. **Неправильно:** полнотекстовый поиск использует синонимы.

6.2 Использование предикатов CONTAINS и FREETEXT

6.2.1 Предикат CONTAINS

Предикат CONTAINS предоставляет возможность выполнять поиск:

- слов и фраз в тексте;
- точных или примерных (fuzzy) совпадений;
- словоформ слова;
- текста, в котором искомое слово находится рядом с другим искомым словом;
- синонимов искомого слова;
- префикса слова или выражения.

6.2.2 Предикат FREETEXT

Предикат FREETEXT является менее определенным и поэтому возвращает большее число строк, чем предикат CONTAINS. Он выполняет поиск значений, совпадающих со смыслом фразы, а не с точным значением слов. Когда вы используете предикат FREETEXT, ядро выполняет разбиение на слова искомой фразы, генерирует словоформы (но не парадигмы) и определяет список расширений и замен для слов в выражениях поиска на основании слов из тезауруса.

Резюме занятия

- Предикат CONTAINS можно использовать для избирательного поиска.
- Предикат FREETEXT можно использовать для более общего поиска.

6.3 Использование табличных функций полнотекстового и семантического поиска

6.3.1 Статистические оконные функции

- SEMANTICKEYPHRASETABLE
- SEMANTICSIMILARITYDETAILSTABLE
- SEMANTICSIMILARITYTABLE

7 Запрос и управление XML-данными

7.1 Возвращение результатов в виде XML с помощью предложения FOR XML

```
1 WITH XMLNAMESPACES('TK461-CustomersOrders' AS co) S
2 ELECT [co:Customer].custid AS [co:custid],
3 [co:Customer].companyname AS [co:companyname],
4 [co:Order].orderid AS [co:orderid],
5 [co:Order].orderdate AS [co:orderdate]
6 FROM Sales.Customers AS [co:Customer]
7 INNER JOIN Sales.Orders AS [co:Order]
8 ON [co:Customer].custid = [co:Order].custid
9 WHERE [co:Customer].custid <= 2
10 AND [co:Order].orderid %2 = 0
11 ORDER BY [co:Customer].custid, [co:Order].orderid
12 FOR XML AUTO, ELEMENTS, ROOT('CustomersOrders');
```

Выводим XML-документ.

```
<CustomersOrders xmlns:co="TK461-CustomersOrders">
  <co:Customer>
    <co:custid>1</co:custid>
    <co:companyname>Customer NRZBB</co:companyname>
    <co:Order>
      <co:orderid>10692</co:orderid>
      <co:orderdate>2007-10-03T00:00:00</co:orderdate>
    </co:Order>
    <co:Order>
      <co:orderid>10702</co:orderid>
      <co:orderdate>2007-10-13T00:00:00</co:orderdate>
    </co:Order>
    <co:Order>
      <co:orderid>10952</co:orderid>
      <co:orderdate>2008-03-16T00:00:00</co:orderdate>
    </co:Order>
  </co:Customer>
  <co:Customer> <co:custid>2</co:custid>
  <co:companyname>Customer MLTDN</co:companyname>
  <co:Order>
    <co:orderid>10308</co:orderid>
    <co:orderdate>2006-09-18T00:00:00</co:orderdate>
  </co:Order>
```

7.1.1 Режим FOR XML PATH

С помощью двух последних возможностей предложения FOR XML — параметров EXPLICIT и PATH — можно вручную определять возвращаемый XML. Используя эти два параметра, вы получаете полный контроль над возвращаемым XMLдокументом.

```
1  SELECT Customer.custid AS [@custid],
2    Customer.companyname AS [companyname]
3    FROM Sales.Customers AS Customer
4    WHERE Customer.custid <= 2
5    ORDER BY Customer.custid
6    FOR XML PATH ('Customer'), ROOT('Customers');
```

```
<Customers>
  <Customer custid="1">
    <companyname>Customer NRZBB</companyname>
  </Customer>
  <Customer custid="2">
    <companyname>Customer MLTDN</companyname>
  </Customer>
</Customers>
```

7.2 Запрос XML-данных с помощью XQuery

```
1  DECLARE @x AS XML;
2  SET @x=N'
3  <root>
4    <a>1<c>3</c><d>4</d></a>
5    <b>2</b>
6  </root>';
7  SELECT
8    @x.query('*') AS Complete_Sequence,
9    @x.query('data(*)') AS Complete_Data,
10   @x.query('data(root/a/c)') AS Element_c_Data;
```

Complete_Sequence	Complete_Data_Element_c_Data
<root><a>1<c>3</c><d>4</d>2</root>	1342
	3

Первое выражение XQuery — простейшее возможное выражение path, которое выбирает все из экземпляра XML; второе использует функцию data() для извлечения атомарных значений данных из всего документа; третье использует функцию data() для извлечения атомарных значений данных только из элемента.

Резюме занятия

- Язык XQuery можно использовать в запросах T-SQL, чтобы запрашивать XMLданные.
- Язык XQuery поддерживает собственные типы данных и функции.
- Выражение XPath используется для навигации по экземпляру XML.
- Настоящая сила XPath заключена в выражениях FLWOR.

8 Создание таблиц и обеспечение целостности данных

8.1 Создание и изменение таблиц

8.1.1 Создание таблицы

В T-SQL можно создать таблицу двумя способами:

- с помощью инструкции CREATE TABLE, где явно определяются компоненты таблицы;
- с помощью инструкции SELECT INTO, которая автоматически создает таблицу, используя выходные данные запроса для основного определения таблицы.

ПРИМЕЧАНИЕ Двухкомпонентное именование таблиц

SQL Server всегда назначает таблице ровно одну схему. Поэтому всегда следует ссылаться на таблицу с использованием двухкомпонентных имен (с указанием имени схемы и таблицы) во избежание ошибок и чтобы сделать код более надежным.

8.1.2 Определение схемы базы данных

Каждая таблица принадлежит к группировке объектов внутри базы данных, которую называют схемой базы данных. Схема базы данных — это поименованный контейнер (пространство имен), который можно использовать для того, чтобы группировать таблицы и другие объекты баз данных.

ВАЖНО!

Схема базы данных и схема таблицы

Не следует путать термин "схема базы данных" с термином "схема таблицы". Схема базы данных — это контейнер объектов в пределах базы данных. Схема таблицы — это определение таблицы, которое включает инструкцию CREATE TABLE со всеми определениями столбцов.

Следующие четыре встроенных схемы баз данных не могут быть удалены:

- схема базы данных dbo — это стандартная (по умолчанию) схема базы данных для новых объектов, созданных пользователями, имеющими роли db_owner или db_ddl_admin;
- схема guest используется для объектов, которые должны быть доступны пользователю guest. Эта схема используется редко;
- схема INFORMATION_SCHEMA используется представлениями Information Schema, которая предоставляет доступ к метаданным по стандарту ANSI;
- схема базы данных sys зарезервирована SQL Server для системных объектов, таких как системные таблицы и представления.

ПРИМЕЧАНИЕ Схемы баз данных не могут иметь вложенности

Возможен только один уровень схемы базы данных; одна схема не может включать в себя другую схему.

1 `CREATE SCHEMA Production AUTHORIZATION dbo;`

Именем схемы Production фактически владеет пользователь с именем dbo, а не схема базы данных dbo. Это позволяет одному пользователю (например, dbo) владеть несколькими разными схемами базы данных.

СОВЕТ

Подготовка к экзамену

Можно переместить таблицу из одной схемы в другую с помощью инструкции ALTER SCHEMA TRANSFER. При условии, что объект с именем Categories не существует в схеме базы данных Sales, следующая инструкция перемещает таблицу production.Categories в схему базы данных Sales.

`ALTER SCHEMA Sales TRANSFER Production.Categories;`

Выполните следующую инструкцию для того, чтобы переместить ее обратно.

`ALTER SCHEMA Production TRANSFER Sales.Categories;`

Можно создать таблицу следующим образом:

1 `CREATE TABLE Production.[Yesterday's News]`

Или это можно записать таким образом:

```
1 CREATE TABLE Production."Tomorrow's Schedule"
```

ПРИМЕЧАНИЕ Регулярные идентификаторы более удобны для пользователей

Хотя квадратные скобки можно применять в качестве разделителей, более правильным считается всегда следить за тем, чтобы эти имена следовали правилам создания регулярных идентификаторов. В этом случае, если пользователь не применяет разделители в запросе, запросы все равно будут работать правильно.

- Типы данных DATE, TIME и DATETIME2 могут хранить данные более эффективно и с большей точностью, чем типы DATETIME и SMALLDATETIME.
- Используйте типы данных VARCHAR(MAX), NVARCHAR(MAX) и VARBINARY вместо устаревших типов данных TEXT, NTEXT и IMAGE.
- Используйте тип данных ROWVERSION вместо устаревшего типа TIMESTAMP.
- Типы DECIMAL и NUMERIC — это один и тот же тип данных, но обычно люди предпочитают DECIMAL, поскольку это имя несколько более описательно. Используйте типы данных DECIMAL и NUMERIC вместо FLOAT или REAL, кроме случаев, когда вам действительно нужна точность с плавающей запятой и известны возможные проблемы с округлением.

```
1 CREATE TABLE Production.Categories(
2     categoryid INT IDENTITY(1,1) NOT NULL,
3     categoryname NVARCHAR(15) NOT NULL,
4     description NVARCHAR(200) NOT NULL DEFAULT ('')
5 ) ON [PRIMARY];
6 GO
```

8.1.3 Свойство идентификатора и порядковые номера

В языке T-SQL свойство идентификатора (identity) может быть назначено столбцу для того, чтобы автоматически генерировать последовательность чисел. Его можно применять только к одному столбцу в таблице, и для генерируемой последовательности номеров нужно указать начальное число (seed) и приращение (increment).

```
1 CREATE TABLE Production.Categories(categoryid INT IDENTITY(1,1) NOT NULL,
... )
```

8.1.4 Сжатие таблиц

Существуют два уровня сжатия данных:

- ROW — при сжатии на уровне строк SQL Server применяет более компактный формат хранения к каждой строке в таблице;
- PAGE — сжатие на уровне страниц включает в себя сжатие на уровне строк плюс дополнительные алгоритмы сжатия, которые могут выполняться на уровне страницы.

Следующая команда добавляет сжатие на уровне строк для таблицы Production как часть инструкции CREATE TABLE.

```
1 CREATE TABLE Sales.OrderDetails
2 ( orderid INT NOT NULL,
3 ... )
4 WITH (DATA_COMPRESSION = ROW);
```

```
1 ALTER TABLE Sales.OrderDetails
2 REBUILD WITH (DATA_COMPRESSION = PAGE);
```

КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Может ли имя таблицы содержать пробелы, апострофы и прочие нестандартные символы?
2. Какие существуют типы сжатия таблиц?

Ответы на контрольные вопросы

1. Да, имена таблиц и столбцов могут быть идентификаторами с разделителями, содержащими нестандартные символы.
2. Можно использовать сжатие страниц или строк в таблицах. Сжатие на уровне страниц включает в себя сжатие на уровне строк.

8.1.5 Изменение таблицы

Нельзя использовать команду ALTER TABLE для:

- изменения имени столбца;
- добавления свойства идентификатора;
- удаления свойства идентификатора.

Резюме занятия

- При создании таблицы задается схема таблицы как пространство имен или контейнер для таблицы.
- Следует соблюдать аккуратность при присвоении имен таблицам и столбцам и делать их описательными.
- Выбирайте наиболее эффективные и точные типы данных для столбцов.
- Выбирайте подходящие свойства для столбцов, такие как свойство столбца IDENTITY, и возможность разрешения в столбце значений NULL.

- Можно задать возможность сжатия таблицы при ее создании.
- Можно использовать команду ALTER TABLE для изменения большинства свойств столбцов уже после создания таблицы.

Закрепление материала

версию или пасхалку, можно паковать в приложение. Отметьте в конце книги.

1. Какие из перечисленных идентификаторов являются регулярными идентификаторами T-SQL? (Выберите все подходящие варианты.)
 - A. categoryname.
 - B. category name.
 - C. category\$name.
 - D. category_name.
2. Какой тип данных следует использовать вместо TIMESTAMP?
 - A. VARBINARY.
 - B. ROWVERSION.
 - C. DATETIME2.
 - D. TIME.
3. Как обозначить, что столбец categoryname разрешает значения NULL?
 - A. categoryname PERMIT NULL NVARCHAR(15).
 - B. categoryname NVARCHAR(15) ALLOW NULL.
 - C. categoryname NVARCHAR(15) PERMIT NULL.
 - D. categoryname NVARCHAR(15) NULL.

Ответы

1. Правильные ответы: А, С и Д.
 - A. **Правильно:** регулярный идентификатор может состоять из всех символов алфавита.
 - B. **Неправильно:** регулярный идентификатор не может включать в себя пробел.
 - C. **Правильно:** регулярный идентификатор может включать в себя знак доллара (\$).
 - D. **Правильно:** регулярный идентификатор может содержать подчеркивание (_).
2. Правильный ответ: В.
 - A. **Неправильно:** тип данных VARBINARY предназначен для хранения двоичных данных общего назначения и не может заменить тип TIMESTAMP.
 - B. **Правильно:** тип данных ROWVERSION заменяет устаревший тип TIMESTAMP.
 - C. **Неправильно:** тип данных DATETIME2 хранит типы данных даты и времени и не может заменить тип TIMESTAMP.

- D. **Неправильно:** тип данных TIME хранит только данные в формате времени и не может заменить тип TIMESTAMP.
3. Правильный ответ: D.
 - A. **Неправильно:** указание NULL должно идти после типа данных.
 - B. **Неправильно:** ALLOW NULL не является допустимой конструкцией инструкции CREATE TABLE.
 - C. **Неправильно:** PERMIT NULL не является допустимой конструкцией инструкции CREATE TABLE..
 - D. **Правильно:** следует указывать NULL сразу после типа данных.

8.2 Обеспечение целостности данных

8.2.1 Использование ограничений

СОВЕТ

Подготовка к экзамену

Поскольку соединения часто выполняются на внешних ключах, можно улучшить производительность запроса созданием некластеризованного индекса на внешнем ключе в ссылающейся таблице. Соответствующий столбец в ссылочной (на которую ссылаются) таблице уже имеет уникальный индекс, но если ссылающаяся таблица, в нашем случае Production.Products, содержит много строк, SQL Server может быстрее разрешить соединение, если он будет иметь возможность использовать индекс на большой таблице.

```
1 CREATE TABLE Production.Products
2   ( productid INT NOT NULL IDENTITY ,
3     productname NVARCHAR(40) NOT NULL ,
4     supplierid INT NOT NULL ,
5     categoryid INT NOT NULL ,
6     unitprice MONEY NOT NULL
7     CONSTRAINT DFT_Products_unitprice DEFAULT(0) ,
8     discontinued BIT NOT NULL
9     CONSTRAINT DFT_Products_discontinued DEFAULT(0) ,
10    ... );
```

Резюме занятия

- Для поддержки целостности данных в таблицах базы данных можно объявить ограничения, которые сохраняются в базе данных.
- Ограничения обеспечивают подчинение данных, введенных в таблицы, более сложным правилам, чем определенные для типов данных и допустимости значений NULL.
- Ограничения таблиц включают в себя ограничения первичного ключа и ограничения уникальности, которые обеспечиваются в SQL Server с помощью уникального индекса. Они также включают ограничения внешнего ключа, гарантирующие, что только данные, правильность которых

проверена в другой таблице уточненных запросов (lookup table), разрешены в исходной таблице. Также к ним относятся проверочные ограничения и ограничения по умолчанию, которые применяются к столбцам.

Закрепление материала

Закрепление материала

Для проверки знания материала, изложенного в данном занятии, ответьте на следующие вопросы. Ответы на эти вопросы и пояснения, почему тот или иной ответ верен или неверен, можно найти в *приложении "Ответы" в конце книги*.

1. Какой из перечисленных столбцов может использоваться в качестве суррогатного ключа? (Выберите все подходящие варианты.)
 - A. Время (в тысячных долях секунды), когда строка была вставлена.
 - B. Автоматически увеличивающееся целое число.
 - C. Последние 4 цифры номера социального страхования, объединенные с первыми 8 знаками фамилии пользователя.
 - D. Уникальный идентификатор (GUID), выбранный из SQL Server в момент вставки строки.
2. Вы хотите гарантировать ввод достоверного значения `supplierid` для каждого значения `productid` в таблице `Production.Products`. Какое ограничение следует использовать?
 - A. Ограничение уникальности.
 - B. Ограничение по умолчанию.
 - C. Ограничение внешнего ключа.
 - D. Ограничение первичного ключа.
3. Какие таблицы метаданных дают возможность получить список ограничений в базе данных? (Выберите все подходящие варианты.)
 - A. `sys.key_constraints`.
 - B. `sys.indexes`.
 - C. `sys.default_constraints`.
 - D. `sys.foreign_keys`.

Ответы

1. Правильные ответы: В и D.

- A. **Неправильно:** суррогатные ключи должны быть незначащими, а время — это значащее число. Кроме того, нельзя гарантировать, что две строки не могут быть добавлены почти в одно и то же время.
- B. **Правильно:** автоматически увеличивающееся целочисленное значение обычно используется в качестве суррогатного ключа, поскольку оно не влияет на значащие данные строки и будет уникальным для каждой строки.
- C. **Неправильно:** суррогатный ключ не должен иметь значащих данных, таких как часть идентификатора пользователя и имя пользователя.
- D. **Правильно:** уникальный идентификатор (GUID) тоже может использоваться как суррогатный ключ, когда он уникальным образом генерируется для каждой строки.

2. Правильный ответ: С.

- A. **Неправильно:** ограничение уникальности только обеспечивает уникальность и не может проверять существование значения в другой таблице.
- B. **Неправильно:** ограничение по умолчанию только задает значение по умолчанию. Оно не может проверять существование значения в другой таблице.
- C. **Правильно:** ограничение внешнего ключа проверяет существование значения в другой таблице.
- D. **Неправильно:** ограничение первичного ключа гарантирует уникальность и не может проверять существование значения в другой таблице.

3. Правильные ответы: А, С и D.

- A. **Правильно:** sys.key_constraints перечисляет все первичные ключи и ограничения уникальности в базе данных.

666

Ответы

- B. **Неправильно:** sys.indexes не перечисляет ограничения уникальности в базе данных.
- C. **Правильно:** sys.default_constraints перечисляет ограничения уникальности в базе данных.
- D. **Правильно:** sys.foreign_keys перечисляет все первичные ключи в базе данных.

Упражнения

Упражнение 1. Работа с ограничениями таблиц

Как ведущий разработчик баз данных на новом проекте, вы заметили, что проверка правильности базы данных выполняется на клиентском приложении. В результате разработчики базы данных периодически запускают очень затратные запросы для проверки целостности данных. Вам нужно принять решение о необходимости рефакторинга базы данных командой для улучшения целостности базы данных и сокращения дорогостоящих, выполняемых каждую ночь, запросов проверки правильности. Ответьте на следующие вопросы о действиях, которые вы можете предпринять.

1. Как можно убедиться, что определенные комбинации столбцов в таблице имеют уникальное значение?
2. Как можно обеспечить ограничение значений в определенных таблицах указанными диапазонами?
3. Как можно гарантировать, что все столбцы, которые содержат значения из таблиц уточняющих запросов, правильны?
4. Как можно гарантировать, что все таблицы имеют первичный ключ, даже те таблицы, которые в данный момент не имеют объявленного первичного ключа?

Упражнение 2. Использование ограничений уникальности и ограничений по умолчанию

При более тщательном обследовании базы данных вашего текущего проекта вы обнаружили, что имеется больше проблем целостности данных, чем вы нашли вначале. Далее перечислены некоторые из обнаруженных вами проблем. Как вы предлагаете их решить?

1. Большинство таблиц имеет суррогатный ключ, который вы реализовали как первичный ключ. Но существуют и другие столбцы или комбинации столбцов, которые должны быть уникальными, и таблица может иметь только один первичный ключ. Как вы можете обеспечить уникальность других конкретных столбцов или комбинаций столбцов?
2. Несколько столбцов разрешают значения `NULL`, хотя предполагается, что приложение всегда их заполняет. Как можно обеспечить, что эти столбцы никогда не будут разрешать значения `NULL`?
3. Часто приложение должно указывать определенные значения для каждого столбца при вставке в строку. Как вы можете настроить столбцы так, что если

Ответы

Упражнения

Упражнение 1. Работа с ограничениями таблиц

1. Можно обеспечить уникальность определенных столбцов или их комбинаций в таблице, применив ограничения первичного ключа и ограничения уникальности. Также можно применить уникальный индекс. Обычно предпочтительно использовать объявленное ограничение первичного ключа и уникальности, поскольку они могут быть легко найдены и распознаны в метаданных и административных инструментах SQL Server. Если уникальность строки не может быть задана с помощью ограничения или уникального индекса, вы можете попробовать применить триггер.
2. Для простых ограничений диапазонов в таблице можно использовать проверочное ограничение. Затем вы можете указать это ограничение в значении выражения ограничения.
3. Для обеспечения правильности искомых значений в основном следует использовать ограничения внешнего ключа. Ограничения внешнего ключа — это объявленные ограничения и поэтому известны SQL Server и оптимизатору запросов через метаданные. При соединении таблицы, которая имеет ограничение внешнего ключа, с ее таблицей уточняющих запросов (lookup table) полезно добавить индекс на столбец внешнего ключа для повышения производительности соединения.
4. Нельзя обеспечить применение ограничения первичного ключа для каждой таблицы. Но вы можете запросить `sys.key_constraints` с целью контроля, что каждая таблица действительно содержит первичный ключ.

Упражнение 2. Использование ограничений уникальности и ограничений по умолчанию

1. Можно создать ограничение уникальности на столбце или наборе столбцов для обеспечения уникальности их значений, в дополнение к первичному ключу.
2. Можно предотвратить наличие значений `NULL` в столбце, изменив таблицу и переопределив столбец как `NOT NULL`.
3. Можно создать ограничение по умолчанию на столбце и, таким образом, обеспечить, что если не вставлены никакие значения, на их место будут вставлены значения по умолчанию.

9 Проектирование и создание представлений, встроенных функций и синонимов

9.1 Проектирование и реализация представлений и встроенных функций

В SQL представления используются для хранения и повторного использования запросов к базе данных. Представления могут использоваться практически для тех же целей, что и таблицы.

```
1 CREATE VIEW Sales.OrderTotalsByYear
2   WITH SCHEMABINDING
3   AS
4     SELECT
5       YEAR(O.orderdate) AS orderyear ,
6       SUM(OD.qty) AS qty
7     FROM Sales.Orders AS O
8     JOIN Sales.OrderDetails AS OD
9     ON OD.orderid = O.orderid
10    GROUP BY YEAR(orderdate);
```

Это представление создано с использованием параметра представления SCHEMABINDING, который гарантирует, что структуры базовых таблиц не могут быть изменены без удаления представления.

ПРИМЕЧАНИЕ Представления показывают пользователю абстрактные уровни

Основное использование представлений в реляционных базах данных, как для оперативной обработки транзакций (online transaction processing, OLTP), так и в системах хранения данных — представить уровень абстракции между конечным пользователем и базой данных. Когда базе данных требуются сложные соединения таблиц, можно упростить запросы пользователя путем внедрения этих соединений в представления. Пользователи запрашивают представления, а не таблицы, что позволяет им работать с более простым, логическим представлением базы данных, без необходимости знать их сложные физические детали.

ПРИМЕЧАНИЕ Создавайте самодокументируемые представления

Считается наиболее правильным создавать самодокументируемый код T-SQL. В общем случае, представление будет более самодокументируемым, если имена столбцов представления указаны в инструкции SELECT, а не перечислены отдельно в представлении.

9.1.1 Параметры представления

Можно добавить следующие параметры представления в любой комбинации.

- С помощью предложения WITH ENCRYPTION можно указать, что текст представления должен быть сохранен в неявном виде (это не строгое шифрование). Это затрудняет для пользователей раскрытие текста инструкции SELECT представления.
- Предложение WITH SCHEMABINDING, как уже говорилось ранее, привязывает представление к схемам базовых таблиц: нельзя изменить определения схем таблицы без удаления представления. Это защищает представление от повреждения в результате изменения структур таблиц.
- Предложение WITH VIEW_METADATA, когда оно указано, возвращает метаданные представления, а не базовой таблицы.

9.1.2 Предложение WITH CHECK OPTION

Предложение WITH CHECK OPTION ограничивает модификации только строками, которые удовлетворяют критериям фильтра.

9.1.3 Ограничения в представлениях

Представления имеют некоторые ограничения, перечисленные далее.

- В представлении нельзя добавить предложение ORDER BY в инструкции SELECT. Представление должно выглядеть точно так же, как таблица, а таблицы в реляционных базах данных содержат наборы строк.

Сами по себе наборы не упорядочены, хотя можно применить к ним сортировку в результирующем наборе с помощью предложения ORDER BY. Аналогично, строки таблиц и представлений в SQL Server не имеют логического порядка, хотя их можно упорядочить, добавив предложение ORDER BY в самой внешней инструкции SELECT при доступе к представлению.

- Представлению нельзя передавать параметры. Аналогично, представление не может ссылаться на переменную внутри инструкции SELECT.
- Представление не может создать таблицу, как постоянную, так и временную. Иными словами, в представлении нельзя использовать синтаксис SELECT/INTO.
- Представление может ссылаться только на постоянные таблицы; оно не может ссылаться на временную таблицу.

СОВЕТ**Подготовка к экзамену**

Результаты представления никогда не упорядочены. Для выборки из представления необходимо добавить свое предложение ORDER BY. Можно включить предложение ORDER BY в представление, только добавив оператор TOP или предложение OFFSET...FETCH в предложении SELECT. Даже тогда результат представления не будет упорядочен. Поэтому присутствие предложения ORDER BY в представлении, даже если вы можете его ввести, бесполезно.

9.1.4 Индексированные представления

Можно создать уникальный кластеризованный индекс на представлении и таким образом материализовать данные. В этом случае хранится не только определение представления. На диске сохраняются фактические результаты запроса, в структуре кластеризованного индекса.

9.1.5 Выполнение запросов из представлений

При выполнении запроса из стандартного нематериализованного представления оптимизатор запросов SQL Server объединяет ваш внешний запрос

с запросом, встроенным в представление, и обрабатывает этот объединенный запрос. В результате, когда вы посмотрите на планы запросов для запросов, в которых выборка выполняется из представлений, то увидите в плане запроса ссылочные базовые таблицы представления; само представление не будет объектом в плане запроса.

9.1.6 Изменение представления

Команда ALTER VIEW просто переопределяет работу представления посредством полного повторного определения представления.

```
1 ALTER VIEW Sales.OrderTotalsByYear
2   WITH SCHEMABINDING
3   AS
4     SELECT O.shipregion,
5        YEAR(O.orderdate) AS orderyear,
6        SUM(OD.qty) AS qty
7     FROM Sales.Orders AS O
8   JOIN Sales.OrderDetails AS OD
9     ON OD.orderid = O.orderid
10    GROUP BY YEAR(orderdate), O.shipregion;
```

9.1.7 Удаление представления

```
1 DROP VIEW Sales.OrderTotalsByYear;
```

9.1.8 Модификация данных с помощью представления

Ограничения:

- DML-инструкции (INSERT, UPDATE и DELETE) должны ссылаться точно на одну таблицу за один раз, неважно, на сколько таблиц ссылается представление.

- Столбцы представления должны напрямую ссылаться на столбцы таблиц и не являться выражениями или функциями, окружающими значение столбца.
- Нельзя модифицировать столбец представления, который является вычисляемым и формируется с помощью операторов UNION/UNION ALL, CROSS JOIN, EXCEPT или INTERSECT.
- Нельзя модифицировать столбец представления, значения которого получены группированием, например, с использованием предложений DISTINCT GROUP BY и HAVING.
- Нельзя модифицировать представление, имеющее конструкции ТОР или OFFSET...FETCH в инструкции SELECT, вместе с предложением WITH CHECK OPTION.

9.1.9 Секционированные представления

Если у вас нет возможности использовать секционирование таблиц, вы можете разделить ваши таблицы вручную и создать представление, которое применяет инструкцию UNION к этим таблицам. Результат этого называется секционированным представлением.

9.1.10 Встроенные функции

```

1 CREATE FUNCTION Sales.fn_OrderTotalsByYear ()
2 RETURNS TABLE
3 AS
4 RETURN
5 ( SELECT
6     YEAR(O.orderdate) AS orderyear ,
7     SUM(OD.qty) AS qty
8     FROM Sales.Orders AS O
9     JOIN Sales.OrderDetails AS OD
10    ON OD.orderid = O.orderid
11    GROUP BY YEAR(orderdate)
12 );

```

КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Должно ли представление состоять только из одной инструкции SELECT?
2. Какие типы представлений доступны в языке T-SQL?

Ответы на контрольные вопросы

1. Технически — да, но можно обойти это ограничение с помощью объединения (используя инструкцию UNION) нескольких инструкций SELECT, которые вместе дадут один результирующий набор.

2. Можно создать стандартные представления, которые являются просто сохраненными инструкциями SELECT, или индексированные представления, которые фактически материализуют данные, в дополнение к секционированным представлениям.

Для создания встроенной табличной функции необходимо выполнить следующие действия.

- Указать параметры. Параметры являются необязательными, но круглые скобки, в которые они должны быть заключены, обязательны.
- Добавить предложение RETURNS TABLE, чтобы проинформировать SQL Server, что это табличная функция.
- После блока AS ввести одну инструкцию RETURN. Она действует как встроенная функция для возвращения внедренной инструкции SELECT.
- Внедрить инструкцию SELECT, которая будет определять, что функция должна возвратить в виде набора строк вызывающей стороне.

```
1 CREATE FUNCTION Sales.fn_OrderTotalsByYear (@orderyear int)
2 RETURNS TABLE
3 AS
4 RETURN
5 ( SELECT orderyear , qty FROM Sales.OrderTotalsByYear
6 WHERE orderyear = @orderyear );
```

Вы можете запросить функцию с передачей года, который хотите видеть, следующим образом:

```
1 | SELECT orderyear, qty FROM Sales.fn_OrderTotalsByYear(2007);
```

9.1.11 Параметры встроенной функции

Встроенные функции имеют два важных параметра, таких же, как и у представления.

- Можно создать функцию с использованием предложения WITH ENCRYPTION затруднив для пользователя раскрытие текста SELECT для функции.
- Можно добавить предложение WITH SCHEMABINDING, которое привязывает схемы таблицы базовых объектов, таких как таблицы или представления, к функции. Схемы объектов, на которые выполняется ссылка, не могут изменяться, пока функция не будет удалена или не будет удален параметр WITH SCHEMABINDING.

Контрольные вопросы

1. Какой тип данных возвращает встроенная функция?
2. Какой тип представления может имитировать встроенная функция?

Ответы на контрольные вопросы

1. Встроенные функции возвращают таблицы и, соответственно, часто называются встроенными функциями, возвращающими табличное значение.
2. Встроенная функция, возвращающая табличное значение, может имитировать параметризованное представление, т. е. представление, которое принимает параметры.

Резюме занятия

- Представления — это сохраненные инструкции SELECT языка T-SQL, которые могут обрабатываться, как если бы это были таблицы.
- Обычно представление состоит только из одной инструкции SELECT, но можно обойти это правило, объединив инструкции SELECT со сходными результатами, используя инструкции UNION или UNION ALL.

- Представления могут ссылаться на несколько таблиц и упростить сложные объединения для пользователей.
- По умолчанию представления не содержат никаких данных. Создание уникального кластеризованного индекса на представлении дает в результате индексированное представление, которое материализует данные.
- При выполнении выборки из представления SQL Server берет самую внешнюю инструкцию SELECT и объединяет ее с инструкцией SELECT из определения представления. Затем SQL Server выполняет эту комбинированную инструкцию SELECT.
- Можно модифицировать данные с помощью представления, но только для одной таблицы за один раз и лишь в столбцах определенных типов.
- Можно добавить к представлению предложение WITH CHECK OPTION для того, чтобы предотвратить любые изменения через представление, которые могут привести к получению в строках значений, не удовлетворяющих предложению WHERE данного представления.
- Представления могут ссылаться на таблицы или представления в других базах данных и на других серверах через связанные серверы.
- Специальные представления, называемые секционированными представлениями, могут быть созданы при условии удовлетворения определенному набору требований, и если SQL Server выполняет маршрутизацию соответствующих запросов и обновлений на нужную секцию представления.
- Встроенные функции можно использовать для имитации параметризованных представлений. Представления языка T-SQL не могут принимать параметры. Но встроенная функция, возвращающая табличное значение, может возвращать те же данные, что и представление, а также принимать параметры, которые могут фильтровать результаты.

Закрепление материала

1. Какие из перечисленных операторов работают в представлениях T-SQL? (Выберите все подходящие варианты.)
 - A. Предложение WHERE.
 - B. Предложение ORDER BY.
 - C. Операторы UNION или UNION ALL.
 - D. Предложение GROUP BY.
2. Каков результат выражения WITH SCHEMABINDING в представлении?
 - A. Представление не может быть изменено без изменения таблицы.
 - B. Таблица, на которую делается ссылка в представлении, не может быть изменена, пока не будет изменена инструкция SELECT представления.
 - C. Таблица, на которую делается ссылка в представлении, не может быть изменена, пока не будет удалено представление.
 - D. Представление не может быть изменено, пока не будет удалена таблица, на которую оно ссылается.

3. Что является результатом предложения WITH CHECK OPTION в представлении, которое имеет предложение WHERE в своей инструкции SELECT?
 - A. Данные больше не могут обновляться через представление.
 - B. Данные могут обновляться через представление, но значения первичного ключа не могут быть изменены.
 - C. Данные могут обновляться через представление, но значения изменяться не могут, что приведет к попаданию строк вне фильтра в предложении WHERE.
 - D. Данные могут обновляться через представление, но только столбцы с проверочными ограничениями могут быть изменены.

Ответы

Закрепление материала

1. Правильные ответы: А, С и D.
 - A. **Правильно:** представление может содержать предложение WHERE.
 - B. **Неправильно:** представление может содержать инструкцию ORDER BY, если используется предложение SELECT TOP, но реальная сортировка результатов не гарантируется.
 - C. **Правильно:** инструкции SELECT можно объединить в представлении с помощью операций UNION и UNION ALL.
 - D. **Правильно:** представление может содержать предложение GROUP BY.
2. Правильный ответ: С.
 - A. **Неправильно:** вы всегда можете изменить представление, не изменения базовой таблицы или таблиц.
 - B. **Неправильно:** если даже вы изменяете представление, в случае применения к представлению предложения WITH SCHEMABINDING, базовая таблица не может быть изменена.
 - C. **Правильно:** предложение WITH SCHEMABINDING означает, что схемы базовой таблицы зафиксированы представлением. Для изменения этой таблицы нужно сначала удалить представление.
 - D. **Неправильно:** никогда не нужно удалять таблицы, чтобы изменить представление.
3. Правильный ответ: С.
 - A. **Неправильно:** предложение WITH CHECK OPTION не препятствует обновлению данных через представление.
 - B. **Неправильно:** предложение WITH CHECK OPTION не ограничивает все обновления только столбцами первичного ключа.
 - C. **Правильно:** назначение предложения WITH CHECK OPTION — предотвратить любые обновления, в результате которых строки могут нарушать правильность предложения WHERE представления. Оно также препятствует обновлению любых строк, находящихся вне действия фильтра предложения WHERE.
 - D. **Неправильно:** предложение WITH CHECK OPTION не имеет никакого отношения к проверочным ограничениям таблицы.

9.2 Использование синонимов

Синонимы — это имена, сохраняемые в базе данных, которые могут использоваться для замещения имен других объектов. Эти имена также действуют в пределах базы данных и добавляются к имени схемы.

9.2.1 Создание синонима

```
1 CREATE SYNONYM dbo.Categories FOR Production.Categories;
```

Затем конечный пользователь может делать выборку из Categories без необходимости указывать схему.

```
1 SELECT categoryid, categoryname, description FROM Categories;
```

Синонимы могут применяться для следующих типов объектов:

- таблицы (включая временные таблицы);
- представления;
- пользовательские функции (скалярные, табличные, встроенные);
- хранимые процедуры (T-SQL, расширенные хранимые процедуры и процедуры фильтра репликации);
- сборки среды CLR (хранимые процедуры, табличные, скалярные и статистические функции).

СОВЕТ**Подготовка к экзамену**

Синонимы не могут ссылаться на другие синонимы. Они могут ссылаться только на такие объекты базы данных, как таблицы, представления, хранимые процедуры и функции. Другими словами, создание цепочки синонимов не разрешается.

ПРИМЕЧАНИЕ Использование инструкции ALTER с синонимами

Нельзя ссылаться на синоним в инструкции DDL, такой как ALTER. Эти инструкции требуют ссылки на базовый объект.

9.2.2 Сравнение синонимов с другими объектами баз данных

Далее приведены преимущества синонимов перед представлениями.

- В отличие от представлений, синонимы могут замещать многие другие типы объектов базы данных, не только таблицы.
- Также как и представления, синонимы могут предоставлять уровень абстракции, давая возможность создать логическое представление системы без необходимости показывать физические имена объектов базы данных конечному пользователю. Если базовый объект изменен, его синоним от этого не пострадает.
- В отличие от представлений, синонимы не способны упростить сложную логику, как, например, представление может упростить сложные соединения. Синонимы в действительности — это просто имена.
- Представление может ссылаться на множество таблиц, а синоним может ссылаться только на один объект.
- Представление может ссылаться на другое представление, тогда как синоним не может ссылаться на другой синоним; создание цепочки синонимов недопустимо.

Когда представление замещает таблицу, пользователь может видеть столбцы и типы данных представления. Но синоним не отображает метаданные базовой таблицы или представления, которые он замещает. Это может быть как преимуществом, так и недостатком, в зависимости от контекста.

- Если вы не хотите предоставлять данные пользователю, это можно рассматривать как преимущество. В среде SSMS, когда пользователь открывает дерево, чтобы взглянуть на синоним, этот пользователь не увидит столбцов или типов данных, если синоним ссылается на таблицу

или представление, а также пользователь не увидит никаких параметров, если синоним ссылается на процедуру или функцию.

- Если вы хотите предоставлять метаданные пользователю как часть его обучения, тогда синоним может быть недостатком. Например, пользователю могла бы понадобиться внешняя документация, чтобы выяснить, какие столбцы доступны.

КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Хранит ли синоним код T-SQL или какие-либо данные?
2. Может ли синоним быть изменен?

Ответы на контрольные вопросы

1. Нет, синоним — это всего лишь имя. Единственное, что сохраняется с синонимом, — это объект, на который нужно ссылаться.
2. Нет, для изменения синонима его нужно удалить и создать заново.

Резюме занятия

- Синоним — это имя, которое ссылается на другие объекты базы данных, такие как таблица, представление, функция или хранимая процедура.
- Синоним не хранит никакого кода T-SQL и никаких данных. Он хранит только объект, на который выполняется ссылка.
- Синонимы находятся в области действия базы данных и поэтому существуют в том же пространстве имен, что и объекты, на которые они ссылаются. Следовательно, синониму нельзя дать то же имя, что и другому объекту базы данных.
- Цепочки синонимов не разрешены; синоним не может ссылаться на другой синоним.
- Синонимы не предоставляют никаких метаданных объектов, на которые они ссылаются.

- Синонимы могут использоваться для предоставления уровня абстракции пользователю с помощью различных имен для объектов баз данных.
- Можно модифицировать данные с помощью синонима, но нельзя изменить базовый объект.
- Чтобы изменить синоним, его надо сначала удалить, а затем снова создать.

Закрепление материала

1. Какие типы объектов баз данных могут иметь синонимы? (Выберите все подходящие варианты.)
 - Хранимые процедуры.
 - Индексы.
 - Временные таблицы.
 - Пользователи базы данных.
2. Какие высказывания из перечисленных ниже справедливы для синонимов? (Выберите все подходящие варианты.)
 - Синонимы не хранят код T-SQL или данные.
 - Синонимам не требуются имена схем.
 - Имена синонимов могут сопоставлять те объекты, на которые они ссылаются.

- D. Синонимы могут ссылаться на объекты в других базах данных или через связанные сервера.
3. Каковы зависимости синонимов и других объектов, на которые они ссылаются?
 - Синонимы могут быть созданы с использованием предложения WITH SCHEMABINDING, чтобы предотвратить изменение базовых объектов.
 - Синонимы могут иметь ссылки на другие синонимы.
 - Синонимы могут быть созданы, чтобы ссылаться на объекты базы данных, которые еще не существуют.
 - Синонимы могут быть изначально созданы без имени схемы, которое можно

Ответы

Закрепление материала

1. Правильные ответы: А и С.

А. **Правильно:** синонимы могут ссылаться на хранимые процедуры.

668 Ответы

В. **Неправильно:** синонимы не могут ссылаться на индексы; индексы не являются объектами базы данных, находящимися в пределах области действия имен схем.

С. **Правильно:** синонимы могут ссылаться на временные таблицы.

Д. **Неправильно:** пользователи базы данных не являются объектами базы данных, находящимися в пределах области действия имен схем.

2. Правильные ответы: А и D.

А. **Правильно:** синонимы — это просто имена, они не хранят код T-SQL или какие-либо данные.

В. **Неправильно:** синонимы — это объекты базы данных, ограниченные областью действия схем базы данных, так же как таблицы, представления, функции и хранимые процедуры, поэтому им нужны имена схем.

С. **Неправильно:** имя синонима (имя схемы плюс имя объекта) не может быть таким же, как у любого другого действующего в пределах схемы объекта базы данных, включая другие синонимы.

Д. **Правильно:** синонимы могут ссылаться на другие объекты базы данных с помощью трехкомпонентных имен и на объекты через связанные серверы, используя имена, состоящие из четырех частей.

3. Правильный ответ: С.

А. **Неправильно:** только представления могут быть созданы с опцией WITH SCHEMABINDING, но не синонимы.

В. **Неправильно:** синонимы не могут ссылаться на другие синонимы; цепочки синонимов не разрешены.

С. **Правильно:** можно создать синоним, который ссылается на несуществующий объект. Но чтобы использовать такой синоним, необходимо обеспечить существование объекта.

Д. **Неправильно:** синонимы всегда требуют имени схемы.

Упражнения

Упражнение 1. Сравнение представлений, встроенных функций и синонимов

Как ведущему разработчику баз данных на новом проекте, вам нужно обеспечить логическое представление базы данных приложениям, которые создают ежедневные отчеты. Ваша задача — подготовить для команды администраторов баз данных отчет, показывающий преимущества и недостатки представлений, встроенных функций и синонимов с точки зрения создания логического представления базы данных. Что бы вы порекомендовали использовать с учетом перечисленных далее условий: представления, встроенные функции или синонимы?

- Разработчики приложения не хотят работать со сложными соединениями при подготовке отчетов. Для обновления данных они предпочитают использовать хранимые процедуры.
- В некоторых случаях вам нужно иметь возможность изменять имена таблиц или представлений без перекодирования приложения.
- В других случаях приложению требуется фильтровать данные отчетов в базе данных с помощью передачи параметров, но разработчики не хотят использовать хранимые процедуры для извлечения этих данных.

Упражнение 2. Преобразование синонимов в другие объекты

Вас только что назначили разработчиком базы данных, которая широко использует синонимы вместо таблиц и представлений. На основании обратной связи от пользователей вам необходимо заменить некоторые синонимы. Какие действия вы мо-

жете предпринять, которые не заставят пользователей или приложения менять код в перечисленных далее случаях?

1. Некоторые синонимы ссылаются на таблицы. Но некоторые таблицы должны фильтроваться. Вам нужно оставить синонимы, но каким-то образом фильтровать возвращаемые таблицами данные.
2. Некоторые синонимы ссылаются на таблицы. Иногда имена столбцов в таблицах могут меняться, но синоним все равно должен возвратить старые имена столбцов.
3. Некоторые синонимы ссылаются на представления. Вам нужно организовать для пользователей возможность видеть имена и типы данных столбцов, возвращаемых представлениями, когда пользователи просматривают базу данных с помощью SSMS.

Ответы

Упражнение 1. Сравнение представлений, встроенных функций и синонимов

- Чтобы избавить разработчиков от необходимости использовать сложные соединения, им можно предоставить представления и встроенные функции, которые скрывают сложность соединений. Поскольку для обновления данных они будут использовать хранимые процедуры, вам не требуется обеспечивать обновляемость представлений.
- Вы можете изменить имена или определения представлений и изменить имена таблиц, не затрагивая приложение, если приложение использует синонимы. Вы должны будете удалить и снова создать синоним при изменении имени базовой

Ответы

669

таблицы или представления, и это должно делаться, когда приложение находится в автономном режиме.

- Вы можете использовать встроенные функции для создания подобных представлению объектов, которые могут фильтроваться по параметрам. Хранимые процедуры не нужны, поскольку пользователи могут ссылаться на встроенные функции из предложения `FROM` запроса.

Упражнение 2. Преобразование синонимов в другие объекты

1. Для фильтрования данных, получаемых из таблицы, можно создать представление или встроенную функцию, которые фильтруют соответствующим образом данные, и заново создать синоним для ссылки на это представление или функцию.
2. Чтобы синонимы работали даже в случае изменения имен столбцов, можно создать представление, которое ссылается на таблицы, и заново создать синоним для ссылки на это представление.
3. Синонимы не могут показывать метаданные. Поэтому при просмотре базы данных в среде SSMS пользователи не увидят имена столбцов и их типы данных под синонимом. Чтобы предоставить пользователям возможность видеть типы данных столбцов базовых таблиц, вы должны заменить синонимы представлениями.

10 Вставка, обновление и удаление данных

10.1 Вставка данных

10.1.1 Инструкция INSERT VALUES

```
1 INSERT INTO Sales.MyOrders(custid, empid, orderdate, shipcountry, freight)
2 VALUES(2, 19, '20120620', N'USA', 30.00);
```

Указание имен целевых столбцов после имени таблицы является необязательным, но считается наиболее правильным.

```
1 INSERT INTO Sales.MyOrders(custid, empid, orderdate, shipcountry, freight)
2 VALUES(3, 17, DEFAULT, N'USA', 30.00);
```

Если значение для столбца не задано, SQL Server сначала проверит, получает ли столбец свое значение автоматически, например, с помощью свойства IDENTITY или ограничения по умолчанию. Если нет, SQL Server проверит, допускается ли в столбце использование значений NULL, и если так, будет присвоено значение NULL. Если и это невозможно, SQL Server сгенерирует ошибку.

10.1.2 Инструкция INSERT SELECT

```
1 INSERT INTO Sales.MyOrders(orderid, custid, empid, orderdate, shipcountry,
2                             freight)
3     SELECT orderid, custid, empid, orderdate, shipcountry, freight
4       FROM Sales.Orders
5      WHERE shipcountry = N'Norway';
```

10.1.3 Инструкция INSERT EXEC

```
1 CREATE PROC Sales.OrdersForCountry @country AS NVARCHAR(15)
2 AS
3     SELECT orderid, custid, empid, orderdate, shipcountry, freight
4     FROM Sales.Orders
5     WHERE shipcountry = @country;
6
7     INSERT INTO Sales.MyOrders(orderid, custid, empid, orderdate, shipcountry,
8         freight)
9     EXEC Sales.OrdersForCountry
@country = N'Portugal';
```

СОВЕТ

Инструкция INSERT EXEC с несколькими запросами

Инструкция INSERT EXEC работает, даже если исходный динамический пакет или хранимая процедура содержат более одного запроса. Но это возможно только в том случае, если все запросы возвращают результирующие наборы, совместимые с определением целевой таблицы.

10.1.4 Инструкция SELECT INTO

Эта инструкция создает целевую таблицу, опираясь на определение источника, и вставляет результирующие строки из запроса в эту таблицу. Кроме самих данных, инструкция копирует из источника некоторые компоненты определения данных, такие как имена столбцов, типы данных, возможность использования значения NULL и свойство IDENTITY.

```
1     SELECT orderid, custid, orderdate, shipcountry, freight
2     INTO Sales.MyOrders
3     FROM Sales.Orders
4     WHERE shipcountry = N'Norway';
```

Исходный столбец orderid имеет свойство IDENTITY, и, соответственно, целевой столбец также определен со свойством IDENTITY. Если вы хотите, чтобы у целевого столбца не было этого свойства, вам следует выполнить некоторую обработку, например, orderid + 0 AS orderid.

сли необходимо, чтобы целевой столбец был определен как недопускающий значения NULL, требуется использовать функцию ISNULL: ISNULL(orderid + 0, -1) AS orderid.

Чтобы преобразовать исходный тип данных столбца orderdate DATETIME в тип данных DATE в целевом столбце и отменить разрешение значений NULL, используйте выражение ISNULL(CAST(orderdate AS DATE), '19000101') AS orderdate.

Помните, что инструкция SELECT INTO не копирует ограничения из исходной таблицы, поэтому, если они нужны, вы сами должны определить их в целевой таблице. Например, следующий код определяет ограничение первичного ключа в целевой таблице.

КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Почему рекомендуется указывать имена целевых столбцов в инструкциях INSERT?
2. В чем заключается разница между инструкциями SELECT INTO и INSERT SELECT?

Ответы на контрольные вопросы

1. Потому что тогда не нужно заботиться о порядке, в котором столбцы определены в таблице. Вы также не пострадаете при изменении порядка столбцов из-за будущего изменения определения, а также при добавлении столбцов, получающих свои значения автоматически.
2. Инструкция SELECT INTO создает целевую таблицу и вставляет в нее результат запроса. Инструкция INSERT SELECT вставляет результат запроса в уже существующую таблицу.

Резюме занятия

- Язык T-SQL поддерживает различные инструкции, выполняющие вставку данных в таблицы в базе данных. К ним относятся инструкции INSERT VALUES, INSERT SELECT, INSERT EXEC, SELECT INTO и др.
- С помощью инструкции INSERT VALUES можно вставлять одну или более строк в целевую таблицу, опираясь на выражения значений.
- С помощью инструкции INSERT SELECT можно вставлять в целевую таблицу результат запроса.

- Можно использовать инструкцию INSERT EXEC для вставки в целевую таблицу результата запросов в динамическом пакете или хранимой процедуре.
- Используя инструкции INSERT VALUES, INSERT SELECT и INSERT EXEC, можно опустить столбцы, получающие свои значения автоматически. Столбец может получить свое значение автоматически, если он имеет связанное с ним ограничение по умолчанию, или свойство IDENTITY, или если он допускает использование значений NULL.
- Инструкция SELECT INTO создает целевую таблицу на основании определения данных в исходном запросе и вставляет результат запроса в целевую таблицу.
- Считается наиболее правильным указывать в инструкции INSERT именно целевых столбцов, чтобы удалить зависимость от порядка следования столбцов в определении целевой таблицы.

Закрепление материала

1. В каком из перечисленных ниже случаев, как правило, нельзя указать целевой столбец в инструкции `INSERT`?
 - A. Если столбец имеет связанное с ним ограничение по умолчанию.
 - B. Если столбец разрешает значения `NULL`.
 - C. Если столбец не разрешает значения `NULL`.
 - D. Если столбец имеет свойство `IDENTITY`.
2. Что именно инструкция `SELECT INTO` не копирует из источника? (Выберите все подходящие варианты.)
 - A. Индексы.
 - B. Ограничения.
 - C. Свойство `IDENTITY`.
 - D. Тrigгеры.
3. В чем заключаются преимущества использования комбинации инструкций `CREATE TABLE` и `INSERT SELECT` перед инструкцией `SELECT INTO`? (Выберите все подходящие варианты.)
 - A. С помощью инструкции `CREATE TABLE` можно контролировать все атрибуты целевой таблицы. Используя инструкцию `SELECT INTO`, невозможно контролировать некоторые атрибуты, такие как целевая файловая группа.

- B. Инструкция `INSERT SELECT` более быстрая по сравнению с инструкцией `SELECT INTO`.
- C. Инструкция `SELECT INTO` блокирует как данные, так и метаданные на весь период транзакции. Это означает, что до окончания транзакции можно столкнуться с блокировкой, связанной и с данными, и с метаданными. Если инструкции `CREATE TABLE` и `INSERT SELECT` запускаются из разных транзакций, блокировки метаданных будут сняты довольно быстро, снижая возможность и продолжительность блокирования метаданных.
- D. Использование инструкций `CREATE TABLE` вместе с `INSERT SELECT` требует написания меньшего количества кода, чем использование инструкции `SELECT INTO`.

Ответы

1. Правильный ответ: D.

- A. **Неправильно:** при желании вы имеете возможность не указывать столбец и позволить ограничению по умолчанию сгенерировать значение, но это не означает, что вы должны пропустить его. Если хотите, вы можете указать собственное значение.
- B. **Неправильно:** еще раз, при желании вы имеете возможность не указывать столбец и позволить SQL Server присвоить значение NULL столбцу, но это не означает, что вы должны пропустить его. Если хотите, вы можете указать собственное значение.
- C. **Неправильно:** если столбец не принимает значения NULL и не получает свое значение автоматически каким-либо способом, вы должны указать его.
- D. **Правильно:** если столбец имеет свойство IDENTITY, обычно нужно пропустить его в инструкции INSERT и предоставить свойству возможность присвоить значение. Чтобы указать собственное значение, следует установить параметр IDENTITY_INSERT, но это, как правило, не используется.

670

Ответы

2. Правильные ответы: A, B и D.

- A. **Правильно:** инструкция SELECT INTO не копирует индексы.
- B. **Правильно:** инструкция SELECT INTO не копирует ограничения.
- C. **Неправильно:** инструкция SELECT INTO копирует свойство IDENTITY.
- D. **Правильно:** инструкция SELECT INTO не копирует триггеры.

3. Правильные ответы: A и C.

- A. **Правильно:** инструкция SELECT INTO поддерживает только ограниченный контроль над определением целевого столбца, в отличие от ее альтернативы, имеющей полный контроль над ним.
- B. **Неправильно:** инструкция INSERT SELECT обычно действует не быстрее, чем инструкция SELECT INTO. В действительности, во многих случаях инструкция SELECT INTO может выигрывать за счет ведения минимального журнала.
- C. **Правильно:** инструкция SELECT INTO блокирует как данные, так и метаданные, и поэтому может вызывать блокирование, связанное с тем и другим. Если инструкции CREATE TABLE и INSERT SELECT выполняются в разных транзакциях, блокировка на метаданных держится только в течение очень короткого периода времени.
- D. **Неправильно:** все наоборот — инструкция SELECT INTO требует меньшего кодирования, поскольку вам не нужно определять целевую таблицу.

10.2 Обновление данных

10.2.1 Инструкция UPDATE

```
1 UPDATE <target table>
2 SET <col 1> = <expression 1>,
3 <col 2> = <expression 2>,
4 ... , <col n> = <expression n>
5 WHERE <predicate>;
```

10.2.2 Обновление с использованием объединения

Стандартный язык SQL не поддерживает объединения в инструкции UPDATE, но язык T-SQL поддерживает.

```
1 UPDATE OD
2 SET OD.discount += 0.05
3 FROM Sales.MyCustomers AS C
4 INNER JOIN Sales.MyOrders AS O
5 ON C.custid = O.custid
6 INNER JOIN Sales.MyOrderDetails AS OD
7 ON O.orderid = OD.orderid
8 WHERE C.country = N'Norway';
```

10.2.3 Инструкция UPDATE и табличные выражения

```
1 WITH C AS
2 (SELECT TGT.custid,
3 TGT.country AS tgt_country, SRC.country AS src_country,
4 TGT.postalcode AS tgt_postalcode, SRC.postalcode AS src_postalcode
5 FROM Sales.MyCustomers AS TGT
6 INNER JOIN Sales.Customers AS SRC
7 ON TGT.custid = SRC.custid )
8 UPDATE C
9 SET tgt_country = src_country,
```

10 `tgt_postalcode = src_postalcode;`

Контрольные вопросы

1. Какие строки таблицы обновляются в инструкции UPDATE без предложения WHERE?
2. Можно ли обновить строки более чем в одной таблице, используя инструкцию UPDATE?

340

Глава 10

Ответы на контрольные вопросы

1. Все строки таблицы.
2. Нет, вы можете использовать в качестве исходных столбцы из нескольких таблиц, но обновить можно только одну таблицу за один раз.

Резюме занятия

- Язык T-SQL поддерживает стандартную инструкцию UPDATE, а также несколько расширений к стандарту.
- Вы можете изменить данные в одной таблице, используя данные другой таблицы, с помощью инструкции UPDATE на основе объединений. Но следует помнить, что если несколько исходных строк соответствуют одной целевой строке, обновление не приведет к ошибке; оно будет недeterminированным. В общем случае следует избегать таких обновлений.
- T-SQL поддерживает обновление данных с помощью табличных выражений. Эта возможность удобна, когда вы хотите иметь возможность увидеть результат запроса до фактического обновления данных. Также это удобно, когда нужно изменить строки с помощью выражений, которые обычно запрещены в предложении SET, например оконных функций.

- Если вам нужно модифицировать строку и запросить результат этого изменения, можно использовать специализированную инструкцию UPDATE с переменной, которая может выполнить это за одно обращение к строке.

Закрепление материала

верен или неверен, можно найти в *приложении Ответы в конце книги*.

1. Как вы можете модифицировать значение столбца в целевой строке и получить результат этой модификации за одно обращение к строке?
 - Используя инструкцию UPDATE на основе соединения.
 - Используя инструкцию UPDATE на основе табличного выражения.
 - Используя инструкцию UPDATE с переменными.
 - Результат невозможно получить с помощью только одного обращения к строке.
2. Какие преимущества имеет использование инструкции UPDATE с помощью соединений? (Выберите все подходящие варианты.)
 - Можно фильтровать строки, чтобы выполнить обновление с использованием информации в связанных строках другой таблицы.
 - Можно обновить несколько таблиц с помощью одной инструкции.
 - Вы можете получить информацию из соответствующих строк одной таблицы, чтобы использовать в исходных выражениях в предложении SET.

- D. Вы можете использовать данные из нескольких исходных строк, которые соответствуют одной целевой строке, для обновления данных в целевой строке.
3. Как можно обновить таблицу, задавая столбцу результат оконной функции?
 - Используя инструкцию UPDATE на основе соединения.
 - Используя инструкцию UPDATE на основе табличного выражения.
 - Используя инструкцию UPDATE с переменными.
 - Цель не может быть достигнута.

Ответы

1. Правильный ответ: С.
 - A. **Неправильно:** поддержка объединений в обновлении не обеспечивает решение задачи за одно обращение к строке.
 - B. **Неправильно:** поддержка обновлений на основе табличных выражений не обеспечивает решение задачи за одно обращение к строке.
 - C. **Правильно:** инструкция UPDATE с использованием переменной может изменить значение столбца, и поместить результат в переменную за одно обращение к строке.
 - D. **Неправильно:** задача может быть решена, как описано в ответе C.
2. Правильные ответы: А и С.
 - A. **Правильно:** объединение может использоваться для фильтрации обновленных данных.
 - B. **Неправильно:** нельзя выполнить обновление нескольких таблиц в одной инструкции UPDATE.
 - C. **Правильно:** объединение дает доступ к информации в других таблицах, которые могут использоваться в исходных выражениях для присвоений.

- D. **Неправильно:** когда несколько исходных строк соответствуют одной целевой строке, получается недетерминированное обновление, в котором используется только одна исходная строка. Кроме того, факт, что такое обновление не завершается ошибкой, должен рассматриваться как недостаток, а не преимущество.
3. Правильный ответ: В.
 - A. **Неправильно:** инструкция UPDATE на основе объединения не может ссылаться на оконные функции в предложении SET.
 - B. **Правильно:** используя инструкцию UPDATE на основе табличного выражения, можно вызвать оконную функцию в списке SELECT внутреннего запроса. Затем вы можете сослаться на псевдоним, который вы присвоили результирующему столбцу во внешнем запросе в предложении SET внешней инструкции UPDATE.
 - C. **Неправильно:** инструкция UPDATE с переменной не может ссылаться на оконные функции в предложении SET.
 - D. **Неправильно:** задача может быть решена, как описано в ответе В.

10.3 Удаление данных

10.3.1 Инструкция DELETE

Инструкция DELETE выполняется с полным протоколированием, и, следовательно, удаление большого количества данных может требовать много времени для своего завершения. Такие объемные удаления могут приводить к значительному увеличению журнала транзакций. Они также могут приводить к укрупнению блокировок, а это означает, что SQL Server укрупняет блокировки мелких фрагментов данных, такие как блокировки строк, до полной блокировки таблицы.

Решением может послужить следующий пример:

```
1 WHILE 1 = 1
2 BEGIN
3     DELETE TOP (1000) FROM Sales.MyOrderDetails
4     WHERE productid = 12;
5     IF @@rowcount < 1000 BREAK;
6 END
```

10.3.2 Инструкция TRUNCATE

Существует несколько важных различий между инструкцией DELETE и инструкцией TRUNCATE.

- Инструкция DELETE записывает в журнал транзакций значительно больше информации, чем инструкция TRUNCATE. В случае инструкции DELETE, SQL Server записывает в журнал реальные данные, которые были удалены. Для инструкции TRUNCATE SQL Server записывает только информацию о том, какие страницы были освобождены. В результате инструкция TRUNCATE выполняется существенно быстрее.
- Инструкция DELETE не пытаетсябросить свойство идентификатора, если оно установлено для столбца в целевой таблице. Инструкция

TRUNCATE делает это. Если вы используете инструкцию TRUNCATE и хотели бы не сбрасывать это свойство, вам необходимо сохранить текущее значение идентификатора в переменной (используя функцию IDENT_CURRENT) и восстановить сохраненное значение свойства после усечения таблицы.

- Инструкция DELETE поддерживается, если имеется внешний ключ, указывающий на запрашиваемую таблицу, при условии, что в ссылающейся таблице нет связанных строк. Инструкция TRUNCATE не разрешена, если внешний ключ указывает на таблицу — даже если в ссылающейся таблице нет связанных строк и даже если внешний ключ запрещен.
- Инструкция DELETE применяется к таблице, являющейся частью индексированного представления. Инструкция TRUNCATE в таком случае является недопустимой.
- Инструкция DELETE требует разрешений DELETE на целевую таблицу. Инструкция TRUNCATE требует разрешений ALTER на целевую таблицу.

Если необходимо удалить все строки таблицы, обычно следует использовать инструкцию TRUNCATE, потому что она значительно быстрее инструкции DELETE. Однако она требует больших разрешений и имеет больше ограничений.

10.3.3 Инструкция DELETE на основе объединений

```
1  DELETE FROM O
2  FROM Sales.MyOrders AS O
3  INNER JOIN Sales.MyCustomers AS C
4  ON O.custid = C.custid
5  WHERE C.country = N'USA';
```

Можно реализовать эту же задачу, используя вложенный запрос объединения, как показано в следующем примере:

```
1  DELETE FROM Sales.MyOrders
2  WHERE EXISTS
3  ( SELECT *
4    FROM Sales.MyCustomers
5    WHERE MyCustomers.custid = MyOrders.custid
6    AND MyCustomers.country = N'USA');
```

Эта инструкция оптимизируется так же, как инструкция, использующая объединение, поэтому, с точки зрения производительности, не существует предпочтения одной версии по сравнению с другой.

10.3.4 Инструкция DELETE с табличными выражениями

```
1  WITH OldestOrders AS
2  ( SELECT TOP (100) *
3    FROM Sales.MyOrders
4    ORDER BY orderdate, ordered )
5  DELETE FROM OldestOrders;
```

КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Какие строки будут удалены из целевой таблицы с помощью инструкции `DELETE`, если не используется предложение `WHERE`?
2. Что может служить альтернативой инструкции `DELETE` без предложения `WHERE`?

Ответы на контрольные вопросы

1. Все строки целевой таблицы.
2. Инструкция `TRUNCATE`. Но между ними существуют различия, которые следует принять во внимание.

Резюме занятия

- С помощью инструкции DELETE можно удалять строки из таблицы и при желании ограничивать удаляемые строки, используя фильтр на основе предиката. Также можно ограничить число удаляемых строк с помощью фильтра ТОР, но тогда не будет возможности контролировать, какие строки выбраны для удаления.
- С помощью инструкции TRUNCATE можно удалять все строки в целевой таблице. Эта инструкция не поддерживает фильтры. Преимущество инструкции TRUNCATE перед инструкцией DELETE заключается в том, что первая использует оптимизированное ведение журнала и поэтому работает быстрее, по сравнению с последней. Однако инструкция TRUNCATE имеет больше ограничений, чем инструкция DELETE, и требует более строгих разрешений.
- T-SQL поддерживает синтаксис DELETE на основе объединений, давая возможность удалять строки из одной таблицы, используя информацию из связанных строк в других таблицах.
- Также язык T-SQL поддерживает удаление строк с помощью табличных выражений, таких как обобщенные табличные выражения (СТЕ) и производные таблицы.

Закрепление материала

1. Как удалить строки из таблицы, для которой вычисление ROW_NUMBER равно 1?
 - A. Вы сошлетесь на функцию ROW_NUMBER в предложении WHERE инструкции DELETE.
 - B. Вы будете использовать табличное выражение, такое как обобщенное табличное выражение или производная таблица, вычисляющее столбец на основе функции ROW_NUMBER, и затем примените инструкцию DELETE с фильтром к табличному выражению.
 - C. Вы будете использовать табличное выражение, такое как обобщенное табличное выражение или производная таблица, вычисляющее столбец на основе функции ROW_NUMBER, и затем примените инструкцию TRUNCATE с фильтром к табличному выражению.
 - D. Задача не имеет решения.
2. Что из нижеперечисленного применимо к инструкции DELETE? (Выберите все подходящие варианты.)
 - A. Эта инструкция записывает больше данных в журнал транзакций, чем инструкция TRUNCATE.
 - B. Эта инструкция сбрасывает свойство IDENTITY.
 - C. Эта инструкция запрещена, если внешний ключ указывает на целевую таблицу.
 - D. Эта инструкция запрещена, если существует индексированное представление на основе целевой таблицы.
3. Что из нижеперечисленного применимо к инструкции TRUNCATE? (Выберите все подходящие варианты.)
 - A. Эта инструкция записывает больше данных в журнал транзакций, чем инструкция DELETE.
 - B. Эта инструкция сбрасывает свойство IDENTITY.
 - C. Эта инструкция запрещена, если внешний ключ указывает на целевую таблицу.
 - D. Эта инструкция запрещена, если существует индексированное представление на основе целевой таблицы.

- B. Эта инструкция сбрасывает свойство IDENTITY.
- C. Эта инструкция запрещена, если внешний ключ указывает на целевую таблицу.
- D. Эта инструкция запрещена, если существует индексированное представление на основе целевой таблицы.

Ответы

Закрепление материала

1. Правильный ответ: В.
 - A. **Неправильно:** нельзя напрямую сослаться на функцию ROW_NUMBER в предложении WHERE инструкции DELETE.
 - B. **Правильно:** используя табличное выражение, можно создать результирующий столбец с помощью функции ROW_NUMBER, а затем сослаться на псевдоним этого столбца в фильтре внешней инструкции.
 - C. **Неправильно:** инструкция TRUNCATE не имеет фильтра.
 - D. **Неправильно:** задача может быть решена, как описано в ответе B.
2. Правильный ответ: А.
 - A. **Правильно:** инструкция DELETE записывает больше информации в журнал транзакций, чем инструкция TRUNCATE.
 - B. **Неправильно:** инструкция DELETE не сбрасывает свойство IDENTITY.
 - C. **Неправильно:** инструкция DELETE разрешена даже тогда, когда имеется внешний ключ, указывающий на таблицу, если нет строк, связанных с удаляемыми.
 - D. **Неправильно:** инструкция DELETE разрешена, когда существует индексированное представление на основе целевой таблицы.
3. Правильные ответы: В, С и D.
 - A. **Неправильно:** инструкция TRUNCATE в отличие от инструкции DELETE использует оптимизированное ведение журнала.

672

Ответы

- B. **Правильно:** инструкция TRUNCATE сбрасывает свойство IDENTITY.
- C. **Правильно:** инструкция TRUNCATE запрещена, когда имеется внешний ключ, указывающий на таблицу.
- D. **Правильно:** инструкция TRUNCATE запрещена, когда существует индексиро-

Упражнения

Упражнение 1. Использование модификаций, поддерживающих оптимизированное ведение журнала

Вы являетесь консультантом в отделе информационных технологий в большой компании розничной торговли. В компании каждую ночь запускается процесс, который сначала удаляет все строки в таблице, используя инструкцию `DELETE`, и затем заполняет эту таблицу данными результата запроса к другим таблицам. Результат содержит несколько десятков миллионов строк. Процесс работает очень медленно. Вас просят дать рекомендации для улучшения ситуации.

1. Дайте рекомендации по усовершенствованию части процесса, выполняющей удаление данных.
2. Дайте рекомендации по усовершенствованию части процесса, выполняющей вставку данных.

Упражнение 2. Усовершенствование процесса обновления данных

Та же компания, которая наняла вас для консультации по поводу неэффективности ночного процесса обработки данных в первом упражнении, снова пригласила вас на работу. Ее сотрудники просят вашего совета в отношении следующего процесса обновления данных.

1. В базе данных имеется таблица, содержащая около 100 млн строк. Около трети существующих строк должно быть обновлено. Можете ли вы дать рекомендации по поводу того, как организовать обновление данных, не вызывая при этом ненужных проблем с производительностью в системе?
2. Имеется инструкция `UPDATE`, которая модифицирует строки в одной таблице, на основе информации из связанных строк в другой таблице. Инструкция `UPDATE` в настоящее время использует отдельный вложенный запрос для каждого столбца, который должен быть модифицирован, получая значение соответствующего столбца из связанной с ним строки в исходной таблице. Инструкция также использует вложенный запрос для фильтрации только строк, которые имеют соот-

Ответы

Упражнение 1. Использование модификаций, поддерживающих оптимизированное ведение журнала

1. Что касается процесса удаления, если необходимо очистить целую таблицу, клиенту стоит рассмотреть использование инструкции TRUNCATE, которая выполняется с минимальным протоколированием.
2. Что касается процесса вставки, он может осуществляться очень медленно, т. к. не использует преимуществ минимального протоколирования. Клиенту нужно оценить целесообразность использования вставки с минимальным протоколированием, например, с помощью инструкции SELECT INTO (что может потребовать сначала удалить целевую таблицу), инструкции INSERT SELECT с параметром TABLELOCK и др. Обратите внимание, должна использоваться простая модель восстановления данных для базы данных или модель восстановления с неполным протоколированием, поэтому клиент должен оценить, приемлемо ли это с точки зрения требований к возможностям восстановления данных, принятых в компании.

Упражнение 2. Усовершенствование процесса обновления данных

1. Клиент должен рассмотреть возможность разработки процесса, который организует выполнение больших обновлений по блокам. Если выполнять обновление в одной большой транзакции, процесс скорее всего приведет к значительному увеличению размера журнала транзакций. Кроме того, такой процесс, вероятно, вызовет укрупнение блокировок, которое может привести к возникновению проблем блокирования.
2. Клиент должен рассмотреть возможность использования инструкции UPDATE на основе объединений вместо существующего применения вложенных запросов. Объем кода значительно уменьшится, а производительность может возрасти. Каждый вложенный запрос требует отдельного обращения к соответствующей строке. Поэтому использование нескольких подзапросов для получения значений из нескольких столбцов приведет к нескольким обращениям к данным. С помощью объединений только за одно обращение к соответствующей строке вы можете получить любое количество значений столбцов, какое вам нужно.

11 Другие виды модификации данных

11.1 Использование объекта последовательности и свойства столбца IDENTITY

При вставке строки в таблицу не нужно указывать значение для столбца IDENTITY, поскольку он получает свое значение автоматически.

В случаях, когда при вставке строк в таблицу вы хотите указать собственные значения для столбца orderid, необходимо установить параметр сеанса, который называется SET IDENTITY_INSERT <table> to ON.

В языке T-SQL имеется несколько функций, которые можно использовать, чтобы запросить последнее сгенерированное значение идентификатора, например, если оно вам необходимо, когда вы выполняете вставку связанных строк в другую таблицу:

- функция SCOPE_IDENTITY возвращает последнее значение идентификатора, сгенерированное в вашем сеансе в данной области;
- функция @@IDENTITY возвращает последнее значение идентификатора, сгенерированное в вашем сеансе независимо от области;
- функция IDENT_CURRENT принимает в качестве входа таблицу и возвращает последнее значение идентификатора, сгенерированное во входной таблице независимо от сессии.

```
1 SELECT
2     SCOPE_IDENTITY() AS SCOPE_IDENTITY ,
3     @@IDENTITY AS [@@IDENTITY] ,
4     IDENT_CURRENT('Sales.MyOrders') AS IDENT_CURRENT;
```

11.1.1 Использование объекта последовательности

В отличие от свойства столбца IDENTITY, последовательность — это независимый объект базы данных. Объект последовательности лишен многих

ограничений, присущих свойству IDENTITY, к которым относятся следующие.

- Свойство IDENTITY привязано к конкретному столбцу в определенной таблице. Вы не можете удалить свойство, уже существующее у столбца, а также добавить это свойство существующему столбцу. Столбец должен быть определен с этим свойством.
- Иногда нужно, чтобы ключи не конфликтовали в разных таблицах, но свойство IDENTITY определяется на уровне таблицы.
- Иногда требуется сгенерировать значение до его использования. Это невозможно при использовании свойства IDENTITY. Необходимо вставить строку и только потом получить новое значение с помощью функции.
- Нельзя обновить столбец IDENTITY.
- Свойство IDENTITY не поддерживает циклическое повторение.
- Инструкция TRUNCATE сбрасывает свойство идентификатора.

Объект последовательности создается как независимый объект базы. Он не привязан к определенному столбцу в определенной таблице. Для создания последовательности можно использовать команду CREATE SEQUENCE. Как минимум, следует указать имя объекта:

```
1 CREATE SEQUENCE <schema>. <object>;
```

Далее приведен пример для определения последовательности, которую можно использовать для генерации идентификаторов заказов.

```
1 CREATE SEQUENCE Sales.SeqOrderIDs AS INT  
2     MINVALUE 1  
3     CYCLE;
```

Чтобы запросить новое значение из последовательности, следует использовать функцию NEXT VALUE FOR <sequence name>. Например, выполните следующий код три раза:

```
1 SELECT NEXT VALUE FOR Sales.SeqOrderIDs;
```

Если вам необходимо изменить текущее значение, сделать это можно, используя следующий код:

```
1 ALTER SEQUENCE Sales.SeqOrderIDs RESTART WITH 1;
```

Далее приведен пример использования функции NEXT VALUE FOR в предложении INSERT VALUES, которое вставляет три строки в таблицу.

```
1 INSERT INTO Sales.MyOrders(orderid, custid, empid, orderdate) VALUES  
2 (NEXT VALUE FOR Sales.SeqOrderIDs, 1, 2, '20120620'),  
3 (NEXT VALUE FOR Sales.SeqOrderIDs, 1, 3, '20120620'),  
4 (NEXT VALUE FOR Sales.SeqOrderIDs, 2, 2, '20120620');
```

Также можно использовать функцию NEXT VALUE FOR в ограничении DEFAULT, что позволяет ограничению автоматически генерировать значения в процессе вставки строк. Чтобы определить такое ограничение DEFAULT для столбца orderid, выполните следующий код:

```
1 ALTER TABLE Sales.MyOrders  
2 ADD CONSTRAINT DFT_MyOrders_orderid  
3 DEFAULT(NEXT VALUE FOR Sales.SeqOrderIDs) FOR orderid;
```

Существует очень большая разница в производительности между использованием NO CACHE и CACHE <some value>.

Если установлен параметр NO CACHE, SQL Server должен делать запись на диск для каждого запроса нового значения последовательности. Но с кэшированием производительность намного выше. По умолчанию в момент написания этого курса значение кэша устанавливалось равным 50.

```
1 ALTER SEQUENCE Sales.SeqOrderIDs  
2 CACHE 100;
```

Возвращаясь к списку ограничений свойства IDENTITY, о которых говорилось ранее, перечислим преимущества объекта последовательности.

- Объект последовательности не привязан к конкретному столбцу в определенной таблице. При желании можно указать новое значение с помощью ограничения DEFAULT. Это ограничение можно добавить к существующему столбцу или удалить у него.
- Поскольку последовательность — это независимый объект базы данных, можно применять ту же последовательность для генерации ключей, которые используются в разных таблицах. Таким образом, не будет конфликта ключей в разных таблицах.
- Можно сгенерировать значение последовательности до его использования, сохраняя результат функции NEXT VALUE FOR в переменной.
- Можно обновлять столбцы с помощью инструкции UPDATE, используя результаты функции NEXT VALUE FOR.
- Объект последовательности поддерживает циклическое повторение.
- Инструкция TRUNCATE не сбрасывает текущее значение объекта последовательности, поскольку последовательность не зависит от использующих ее таблиц.

Контрольные вопросы

1. Сколько столбцов, имеющих свойство IDENTITY, может быть в таблице?
2. Как можно получить новое значение из последовательности?

Ответы на контрольные вопросы

1. Один.
2. С помощью функции NEXT VALUE FOR.

Резюме занятия

- SQL Server поддерживает две возможности, позволяющие генерировать последовательность ключей: свойство столбца IDENTITY и объект последовательности.

- Свойство столбца IDENTITY определяется с начальным значением и приращением. При вставке новой строки в целевую таблицу вам не нужно указывать значение для столбца IDENTITY, вместо этого SQL Server генерирует его автоматически.
- Чтобы получить вновь сгенерированное свойство идентификатора, можно выполнить запрос с помощью функций COPE_IDENTITY, @@IDENTITY и IDENT_CURRENT. Первая возвращает последнее значение идентификатора, сгенерированное в данном сеансе и диапазоне. Вторая возвращает последний идентификатор, сгенерированный в данном сеансе. Третья возвращает последний идентификатор, сгенерированный во входной таблице.
- Объект последовательности — это независимый объект в базе данных. Он не привязан к определенному столбцу в конкретной таблице.
- Объект последовательности поддерживает определение начального значения, значения приращения, минимального и максимального поддерживаемых значений, циклического повторения и кэширования.
- Функция NEXT VALUE FOR используется для запроса нового значения из последовательности. Эту функцию можно применять в инструкциях INSERT и UPDATE, ограничениях DEFAULT и присвоениях значений переменным.
- Объект последовательности обходит многие ограничения свойства IDENTITY.

Закрепление материала

1. Какую функцию вы будете использовать для возвращения последнего значения идентификатора, сгенерированного в определенной таблице?
 - A. MAX.
 - B. SCOPE_IDENTITY.
 - C. @@IDENTITY.
 - D. IDENT_CURRENT.
2. В чем заключаются преимущества использования объекта последовательности вместо свойства IDENTITY? (Выберите все подходящие варианты.)
 - A. Свойство IDENTITY не гарантирует отсутствие разрывов, а объект последовательности гарантирует.
 - B. Свойство IDENTITY не может быть добавлено или удалено для существующего столбца; ограничение DEFAULT с функцией NEXT VALUE FOR может быть добавлено или удалено для существующего столбца.
 - C. Новое значение идентификатора не может быть сгенерировано перед использованием инструкции INSERT, тогда как значение последовательности может.
 - D. Вы не можете задать собственное значение, когда выполняете вставку строки в таблицу со столбцом IDENTITY, без специальных разрешений. Но вы можете указать собственное значение для столбца, который обычно получает свое значение из объекта последовательности.

3. Как вы сгенерируете значения последовательности в определенном порядке в инструкции INSERT SELECT?
 - A. Используйте предложение OVER в функции NEXT VALUE FOR.
 - B. Укажите предложение ORDER BY в конце запроса.
 - C. Используйте в запросе параметр TOP (100) PERCENT и предложение ORDER BY.
 - D. Используйте в запросе параметр TOP (9223372036854775807) PERCENT и предложение ORDER BY.

Ответы

Закрепление материала

1. Правильный ответ: D.
 - A. **Неправильно:** максимальное значение в таблице — это не обязательно последнее сгенерированное значение идентификатора.
 - B. **Неправильно:** функция SCOPE_IDENTITY определяется не на уровне таблицы, а на уровне сеанса и области.
 - C. **Неправильно:** функция @@IDENTITY определяется не на уровне таблицы, а на уровне сеанса.
 - D. **Правильно:** функция IDENT_CURRENT принимает имя таблицы на вход и возвращает последнее значение идентификатора, сгенерированное в таблице.
2. Правильные ответы: B, C и D.
 - A. **Неправильно:** оба не могут гарантировать отсутствие разрывов.
 - B. **Правильно:** одно из преимуществ использования объекта последовательности вместо свойства IDENTITY — это то, что можно присоединить ограничение DEFAULT, которое имеет вызов функции NEXT VALUE FOR к существующему столбцу, а также удалить это ограничение для столбца.
 - C. **Правильно:** можно сгенерировать новое значение последовательности до его использования с помощью присвоения этого значения переменной и последующего использования этой переменной в инструкции INSERT.
 - D. **Правильно:** вы можете указать собственное значение для столбца, имеющего свойство IDENTITY, но для этого требуется включить параметр сеанса IDENTITY_INSERT, который в свою очередь требует дополнительных разрешений. Объект последовательности является более гибким. Вы можете вставлять свои значения в столбец, который обычно получает их из объекта последовательности. И это — без необходимости включать какие-либо специальные параметры или предоставлять дополнительные разрешения.
3. Правильный ответ: A.
 - A. **Правильно:** с помощью предложения OVER можно контролировать порядок задания значений последовательности в инструкции INSERT SELECT.
 - B. **Неправильно:** добавление предложения ORDER BY в конец запроса не обеспечивает того, что значения последовательности будут сгенерированы в том же порядке.
 - C. **Неправильно:** использование параметра TOP с предложением ORDER BY не обеспечивает того, что значения последовательности будут сгенерированы в том же порядке.

11.2 Слияние данных

11.2.1 Использование инструкции MERGE

С помощью инструкции MERGE выполняется слияние данных из исходной таблицы или табличного выражения и целевой таблицы с помещением их в целевую таблицу. Общий формат инструкции MERGE выглядит следующим образом:

```
1 MERGE INTO <target table> AS TGT
2 USING <SOURCE TABLE> AS SRC
3 ON <merge predicate>
4 WHEN MATCHED [AND <predicate>]
5 THEN <action>
6 WHEN NOT MATCHED [BY TARGET] [AND <predicate>]
7
8 THEN INSERT...
9 WHEN NOT MATCHED BY SOURCE [AND <predicate>]
10 THEN <action>;
```

Далее перечислены предложения этой инструкции и их роли.

- MERGE INTO <target table>. Это предложение определяет целевую таблицу (target table) данной операции. При желании можно задать псевдоним таблицы в этой операции.
- USING <source table>. Это предложение определяет исходную таблицу (source table) для данной операции
- ON <merge predicate>. В этом предложении вы указываете предикат (merge predicate), который выполняет сопоставление строк между источником и целевой таблицей и определяет, имеется или нет целевая строка, соответствующая исходной строке. Обратите внимание, это предложение не является фильтром, как, например, предложение ON в соединении.
- WHEN MATCHED [AND <predicate>] THEN <action>. Это предложение определяет действие (action), которое следует предпринять, если строке источника соответствует целевая строка.

ВАЖНО!**Избегайте конфликтов слияния**

Предположим, что некоторый ключ K пока что не существует в целевой таблице. Два процесса, P1 и P2, выполняют инструкцию MERGE, подобную предыдущей, в одно и то же время и с одним и тем же исходным ключом K. Инструкция MERGE, принадлежащая процессу P1, вполне может вставить новую строку с ключом K между моментами времени, когда инструкция MERGE, принадлежащая P2, проверяет, имеет ли целевая строка этот ключ, и вставляет строку. В таком случае инструкция MERGE, принадлежащая P2, завершится сбоем из-за нарушения ограничения первичного ключа. Чтобы предотвратить такую ситуацию, используйте подсказку SERIALIZABLE или HOLDLOCK (они имеют эквивалентное значение) к целевой таблице, как показано в предыдущей инструкции. Подробнее уровень изоляции SERIALIZABLE рассмотрен в главе 12.

СОВЕТ**Инструкции MERGE****минимально требуется только одно предложение**

Инструкция MERGE не всегда требует присутствия предложений WHEN MATCHED и WHEN NOT MATCHED; в минимальном варианте вы должны указать только одно предложение, и это может быть любое из трех предложений WHEN. Например, инструкция MERGE, которая указывает только предложение WHEN MATCHED, является стандартной альтернативой инструкции UPDATE, основанной на соединении, которая не является стандартной.

ВАЖНО!**Предикат MERGE и значения NULL**

При проверке, отличается ли значение целевого столбца от значения исходного столбца, предикат инструкции MERGE использует оператор неравенства (\neq). В данном примере ни целевой, ни исходный столбцы не могут иметь значение NULL. Но если использование зна-

чений NULL разрешено в данных, необходимо добавить логику для их обработки и рассмотреть случай, когда одна сторона имеет значение NULL, а другая — нет. Например, пусть столбец `custid` разрешает значения NULL. Вам нужно использовать следующий предикат:

Резюме занятия

- С помощью инструкции MERGE можно выполнить слияние данных из исходной таблицы или табличного выражения в целевую таблицу.

- Целевую таблицу указывают в предложении MERGE INTO, а исходную таблицу — в предложении USING. Предложение USING похоже на предложение FROM инструкции SELECT, что означает возможность использования табличных операторов, табличных выражений, табличных функций и пр.
- Предикат слияния указывается в предложении ON, которое определяет, сопоставляется ли строке источника целевая строка и сопоставляется ли целевой строке строка источника. Помните, что предложение ON не используется для фильтрации данных; напротив, оно служит только для определения соответствия или несоответствия строк и для определения действий, которые следует применить к целевой таблице.
- Должны быть определены разные предложения WHEN, которые указывают, какое действие применить к целевой таблице в зависимости от выходных данных предиката. Можно указать действия, которые следует выполнить, когда исходной строке соответствует целевая строка, когда исходной строке не соответствует ни одна целевая строка и когда целевой строке не соответствует исходная строка.

Закрепление материала

верен или неверен, можно найти в *приложении "Ответы" в конце книги*.

1. Какие предложения WHEN минимально требуются в инструкции MERGE?
 - A. Как минимум, требуются предложения WHEN MATCHED и WHEN NOT MATCHED.
 - B. Как минимум, требуется одно предложение, и это может быть любое предложение WHEN.
 - C. Как минимум, требуется предложение WHEN MATCHED.
 - D. Как минимум, требуется предложение WHEN NOT MATCHED.
2. Что может быть указано в качестве исходных данных в предложении USING? (Выберите все подходящие варианты.)
 - A. Обычная таблица, табличная переменная или временная таблица.
 - B. Табличное выражение, такое как производная таблица или обобщенное табличное выражение.
 - C. Хранимая процедура.
 - D. Табличная функция, такая как OPENROWSET или OPENXML.
3. Какое предложение инструкции MERGE не является стандартным?
 - A. Предложение WHEN MATCHED.
 - B. Предложение WHEN NOT MATCHED.

- C. Предложение WHEN NOT MATCHED BY SOURCE.
- D. Все предложения MERGE являются стандартными.

Ответы

Закрепление материала

1. Правильный ответ: В.
 - A. **Неправильно:** требуется только одно предложение как минимум.
 - B. **Правильно:** требуется только одно предложение как минимум, и это может быть любое из предложений WHEN.
 - C. **Неправильно:** не существует специального предложения WHEN, которое необходимо; как минимум требуется одно любое предложение.
 - D. **Неправильно:** не существует специального предложения WHEN, которое необходимо; как минимум требуется одно любое предложение.
2. Правильные ответы: А, В и D.
 - A. **Правильно:** разрешены таблицы, табличные переменные и временные таблицы.
 - B. **Правильно:** табличные выражения разрешены.
 - C. **Неправильно:** хранимые процедуры не разрешены в качестве источника в инструкции MERGE.
 - D. **Правильно:** табличные функции разрешены.
3. Правильный ответ: С.
 - A. **Неправильно:** предложение WHEN MATCHED является стандартным.
 - B. **Неправильно:** предложение WHEN NOT MATCHED является стандартным.
 - C. **Правильно:** предложение WHEN NOT MATCHED BY SOURCE не является стандартным.
 - D. **Неправильно:** предложение WHEN NOT MATCHED BY SOURCE не является стандартным.

11.3 Использование предложения OUTPUT

11.3.1 Инструкция INSERT с предложением OUTPUT

Предложение OUTPUT может использоваться в инструкции INSERT для того, чтобы возвращать информацию из вставленных строк. Примером его практического использования может служить многострочная инструкция INSERT которая генерирует новые ключи с помощью свойства IDENTITY или последовательности, а вам нужно знать, какие новые ключи были сгенерированы.

1 | `INSERT INTO Sales.MyOrders(custid, empid, orderdate)`

```
1 OUTPUT
2 inserted.orderid, inserted.custid, inserted.empid, inserted.orderdate
3 SELECT custid, empid, orderdate
4 FROM Sales.Orders
5 WHERE shipcountry = N'Norway';
```

Если вам нужно сохранить результат в таблице, а не возвращать его назад вызывающей стороне, добавьте предложение INTO с именем целевой таблицы следующим образом:

```
1 INSERT INTO Sales.MyOrders(custid, empid, orderdate)
2 OUTPUT
3 inserted.orderid, inserted.custid, inserted.empid, inserted.orderdate
4 INTO SomeTable(orderid, custid, empid, orderdate)
5 SELECT custid, empid, orderdate
6 FROM Sales.Orders
7 WHERE shipcountry = N'Norway';
```

11.3.2 Инструкция DELETE с предложением OUTPUT

```
1 DELETE FROM Sales.MyOrders
2 OUTPUT deleted.orderid
3 WHERE empid = 1;
```

11.3.3 Инструкция UPDATE с предложением OUTPUT

```
1 UPDATE Sales.MyOrders
2 SET orderdate = DATEADD(day, 1, orderdate)
3 OUTPUT inserted.orderid,
4 deleted.orderdate AS old_orderdate,
5 inserted.orderdate AS neworderdate
6 WHERE empid = 7;
```

11.3.4 Инструкция MERGE с предложением OUTPUT

SQL Server предоставляет функцию action. Эта функция возвращает строку ('INSERT', 'UPDATE' или 'DELETE'), обозначающую выполненную операцию.

```
1 MERGE INTO Sales.MyOrders AS TGT
2 USING (VALUES(1, 70, 1, '20061218')
3           (2, 70, 7, '20070429'))
4     AS SRC(orderid, custid, empid, orderdate)
5   ON SRC.orderid = TGT.orderid
6 WHEN MATCHED AND (TGT.custid <> SRC.custid
7 OR TGT.empid <> SRC.empid
8 OR TGT.orderdate <> SRC.orderdate) THEN UPDATE
9   SET TGT.custid = SRC.custid,
10      TGT.empid = SRC.empid,
11      TGT.orderdate = SRC.orderdate
12 WHEN NOT MATCHED THEN INSERT
13   VALUES(SRC.orderid, SRC.custid, SRC.empid, SRC.orderdate)
14 WHEN NOT MATCHED BY SOURCE THEN
15   DELETE
16 OUTPUT
17   $action AS the_action,
18   COALESCE(inserted.orderid, deleted.orderid) AS orderid;
```

COBET

MERGE И OUTPUT

В инструкциях INSERT, UPDATE и DELETE можно ссылаться только на столбцы из целевой таблицы в предложении OUTPUT. В инструкции MERGE можно ссылаться на столбцы как целевой таблицы, так и исходной таблицы.

11.3.5 Компонуемый DML

В качестве примера компонуемого DML рассмотрим предыдущую инструкцию MERGE . Предположим, что вам нужно получить лишь строки, к которым применялась только операция INSERT, и передать их в табличную переменную для последующей обработки. Этого можно достичнуть с помощью следующего кода:

```
1 SELECT orderid, custid, empid, orderdate
2 FROM (MERGE INTO Sales.MyOrders AS TGT
3 ...
4 ...
5 ...
6 ...
7 OUTPUT
8     $action AS the_action, inserted.*)
9 WHERE the_action = 'INSERT';
```

КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Сколько предложений OUTPUT может содержать одна инструкция?
2. Как можно определить, какое действие было выполнено применительно к строке OUTPUT в инструкции MERGE?

Ответы на контрольные вопросы

1. Два — одно с INTO и одно без INTO.
2. Используйте функцию \$action.

Резюме занятия

- С помощью предложения OUTPUT можно возвращать информацию из измененных строк в инструкциях модификации.
- Предложение OUTPUT построено как предложение SELECT, что позволяет формировать выражения и присваивать результирующим столбцам псевдонимы.
- Результат предложения OUTPUT можно отправить обратно вызывающей стороне в виде результирующего набора из запроса или сохранить в целевой таблице с помощью предложения INTO.
- При ссылке на столбцы из модифицированных строк следует присваивать в качестве префикса именам столбцов ключевое слово inserted для вставленных строк и deleted для удаленных строк.

- В инструкции MERGE можно использовать функцию action для возвращения строки, которая представляет действие, примененное к целевой строке.
- Компонуемый DML следует использовать для фильтрации выходных строк, которые необходимо сохранить в целевой таблице.

Закрепление материала

1. Когда следует предварять имена столбцов префиксом в виде ключевого слова `inserted` при ссылке в предложении `OUTPUT` на столбцы из вставленных строк?
 - Всегда.
 - Никогда.
 - Только когда используется инструкция `UPDATE`.
 - Только когда используется инструкция `MERGE`.
2. Что является ограничением для таблицы, указанной в качестве целевой в предложении `OUTPUT INTO`? (Выберите все подходящие варианты.)
 - Таблица может быть только табличной переменной.
 - Таблица может быть только временной таблицей.
 - Таблица не может быть задействована ни в одной части отношения по внешнему ключу.
 - Таблица не может иметь триггеров, определенных на ней.
3. Что из нижеперечисленного возможно только при использовании инструкции `MERGE` с точки зрения применения предложения `OUTPUT`?
 - Ссылка на столбцы из исходной таблицы.
 - Ссылка и на ключевое слово `deleted`, и на ключевое слово `inserted`.
 - Назначение псевдонимов выходным столбцам.
 - Использование компонуемого DML.

Ответы

1. Правильный ответ: А.

- А. **Правильно:** при ссылке на элементы из вставленных строк всегда необходимо использовать ключевое слово `inserted` как префикс к имени столбца.
- В. **Неправильно:** не существует случаев, когда можно опустить ключевое слово `inserted`, даже если это инструкция `INSERT`.

Ответы

675

- С. **Неправильно:** верно то, что вы должны добавлять префикс в виде ключевого слова `inserted` к вставленным элементам в инструкции `UPDATE`, но не только в инструкции `UPDATE`.
- Д. **Неправильно:** верно то, что вы должны добавлять префикс в виде ключевого слова `inserted` к вставленным элементам в инструкции `MERGE`, но не только в инструкции `MERGE`.
2. Правильные ответы: С и D.
- А. **Неправильно:** также поддерживаются другие типы таблиц.
- Б. **Неправильно:** также поддерживаются другие типы таблиц.
- С. **Правильно:** целевая таблица не может участвовать в отношении по внешнему ключу.
- Д. **Правильно:** целевая таблица не может иметь триггеры, определенные на ней.
3. Правильный ответ: А.
- А. **Правильно:** на элементы из исходной таблицы можно ссылаться только в предложении `OUTPUT` инструкции `MERGE`.
- Б. **Неправильно:** это также можно сделать в инструкции `UPDATE`.
- С. **Неправильно:** присвоение псевдонимов целевым столбцам в предложении `OUTPUT` разрешено во всех инструкциях.
- Д. **Неправильно:** компонуемый DML поддерживает все инструкции.

Упражнения

Упражнение 1. Лучшее решение для генерации ключей

Вы являетесь членом группы администраторов баз данных в компании, занимающейся производством туристского снаряжения. Большинство таблиц в базах данных OLTP компании в настоящий момент использует свойство `IDENTITY`, но требует большей гибкости. Например, часто приложению нужно генерировать новый ключ до его использования. Иногда приложению требуется обновить ключевой столбец, перезаписав в него новые значения. Кроме того, предложению требуется генерировать ключи, которые не конфликтуют при использовании в разных таблицах.

1. Предложите альтернативу использованию свойства столбца `IDENTITY`.
2. Объясните, как ваше альтернативное предложение может решить существующие проблемы.

Упражнение 2. Усовершенствование модификаций

Вы работаете в группе поддержки баз данных в компании, которая недавно перешла с версии SQL Server 2000 на версию SQL Server 2005 и затем на версию SQL Server 2012. До сих пор используется код, совместимый с SQL Server 2000. Существуют проблемы с модификациями, внесеннымими приложением в базу данных.

Приложение использует процедуру, которая принимает в качестве входных данных атрибуты строки. Затем процедура использует логику, которая проверяет, существует ли ключ в целевой таблице, и при его наличии обновляет целевую строку. В противном случае процедура вставляет новую строку в целевую таблицу. Проблема заключается в том, что периодически процедура дает сбой из-за нарушения ограничения первичного ключа. Это происходит, когда проверка на наличие строки не находит ее, но между этой проверкой и вставкой кто-то еще успевает вставить новую строку с тем же ключом.

В приложении имеется ежемесячный процесс, который архивирует данные, подлежащие удалению. В настоящий момент приложение сначала копирует данные, которые должны быть удалены, в архивную таблицу в одной инструкции и затем удаляет эти строки в другой инструкции. Обе инструкции используют фильтр на основе столбца с датой, который имеет имя `dt`. Требуется отфильтровывать строки, в которых `dt` меньше определенной даты. Проблема заключается в том, что иногда строки, представляющие последние прибытия, вставляются в таблицу между копированием и удалением строк, и процесс удаления заканчивается удалением строк, которые не были заархивированы.

Перед вами поставлена задача найти решения существующих проблем.

1. Можете ли вы предложить решение существующей проблемы с процедурой, которая обновляет строку, когда исходный ключ существует в целевой таблице, и вставляет строку, если его нет?
2. Можете ли вы предложить решение проблемы с процессом архивирования, которое предотвратит удаление строк, которые еще не заархивированы?

Ответы

Упражнение 1. Лучшее решение для генерации ключей

1. Все существующие проблемы, связанные со свойством `IDENTITY`, можно решить с помощью объекта последовательности.
2. С помощью объекта последовательности можно генерировать значения до их использования, с помощью вызова функции `NEXT VALUE FOR` и сохранения результата в переменной. В отличие от свойства `IDENTITY`, вы можете обновить столбец, который обычно получает свои значения из объекта последовательности. Также, поскольку объект последовательности не привязан ни к какому конкретному столбцу в определенной таблице, а напротив, является независимым объектом базы данных, вы можете генерировать значения из одной последовательности и использовать их в различных таблицах.

Упражнение 2. Усовершенствование модификаций

1. Рекомендуемое решение — использовать инструкцию `MERGE`. Определите источник для инструкции `MERGE` как производную таблицу на основе предложения `VALUES`, со строкой, составленной из входных параметров для процедуры. Укажите табличную подсказку `HOLDLOCK` или `SERIALIZABLE` по отношению к целевой таблице, чтобы предотвратить такие конфликты, как существующие в системе

676

Ответы

в данный момент. Затем используйте предложение `WHEN MATCHED` для выполнения действия `UPDATE`, если целевая строка существует, и предложение `WHEN NOT MATCHED`, чтобы выполнить действие `INSERT`, если целевая строка отсутствует.

2. Одна из возможностей — использовать уровень изоляции `SERIALIZABLE`, обрабатывая и инструкцию, которая копирует строки в архив, и инструкцию, которая удаляет строки, в одной транзакции. Но более простым решением является выполнение обеих задач в одной инструкции — инструкции `DELETE` с предложением `OUTPUT INTO`. Это гарантирует, что только строки, которые скопированы в архивную таблицу, будут удалены. И если по какой-то причине произойдет сбой копирования строк в архивную таблицу, операция удаления тоже потерпит неудачу, поскольку оба действия являются частями одной транзакции.

12 Реализация транзакций, обработка ошибок и динамический SQL

12.1 Управление транзакциями и параллелизм

Транзакция — это логическая единица работы. Либо вся работа выполняется как единое целое, либо она не выполняется вовсе. В SQL Server все изменения данных в базе данных происходят в виде транзакций.

12.1.1 Свойства транзакций ACID

- **Атомарность** (atomicity). Каждая транзакция является атомарной единицей работы. Это означает, что в транзакции либо выполняются все изменения базы данных, либо ни одно из них.
- **Согласованность** (consistency). Каждая транзакция, как завершившаяся, так и прерванная, оставляет базу данных в согласованном состоянии, как определяется всеми ограничениями объектов и базы данных. При возникновении несогласованного состояния SQL Server выполнит откат транзакции, чтобы поддержать согласованное состояние.
- **Изоляция** (isolation). Каждая транзакция выполняется так, как будто она существует в изоляции от всех остальных транзакций по отношению к изменениям в базе данных. Степень изолированности может меняться в зависимости от уровня изоляции.
- **Устойчивость** (durability). Каждая транзакция претерпевает прерывание сервиса. Когда сервис восстанавливается, все зафиксированные (committed) транзакции накатываются (зафиксированные изменения в базе данных завершены), а все нефиксированные транзакции откатываются (нефиксированные изменения удаляются).

SQL Server поддерживает транзакционную устойчивость с помощью журнала транзакций базы данных. Любое изменение базы данных (инструкция

модификации данных или инструкция DDL) сначала записывается в журнал транзакций, с исходной версией данных (в случае обновлений и удалений). Когда транзакция зафиксирована и прошли все проверки согласованности, факт успешной фиксации транзакции записывается в журнал транзакций.

Контрольные вопросы

1. Почему для SQL Server важно поддерживать качество транзакций, соответствующее свойствам ACID?
2. Как SQL Server реализует устойчивость транзакций?

396

Глава 12

Ответы на контрольные вопросы

1. Чтобы гарантировать, что целостность данных в базе данных не будет нарушена.
2. Записывая все изменения в журнал транзакций базы данных до внесения изменений в данные базы данных.

12.1.2 Типы транзакций

- **Системные транзакции.** SQL Server использует системные транзакции для поддержки всех своих внутренних постоянных системных таблиц. Пользователи не могут управлять этими транзакциями.
- **Пользовательские транзакции.** Транзакции, создаваемые пользователями в процессе изменения или даже чтения данных, либо автоматически, т. е. неявно, либо явно, называются пользовательскими транзакциями.

12.1.3 Команды транзакций

- BEGIN TRANSACTION (BEGIN TRAN)
- COMMIT TRANSACTION (COMMIT TRAN) (COMMIT WORK) (COMMIT)
- ROLLBACK TRANSACTION (ROLLBACK)

12.1.4 Уровни и состояния транзакций

Уровень транзакции и ее состояние можно определить с помощью двух системных функций.

- Функцию @@TRANCOUNT можно запросить для того, чтобы узнать уровень вложенности транзакции. Уровень, равный 0, указывает, что в данный момент код находится не внутри транзакции. Уровень больше 0 указывает, что транзакция активна, при этом если значение больше 1, то оно указывает уровень вложенности для вложенных транзакций.
- Функцию XACT_STATE() можно запросить для определения состояния транзакции. Состояние, равное 0, указывает на то, что это неактивная транзакция. Состояние, равное 1, указывает, что это незафиксированная транзакция, которая может быть зафиксирована, но не известен уровень ее вложенности. Состояние, равное -1, указывает, что имеется незафиксированная транзакция, но она не может быть зафиксирована из-за предшествующей фатальной ошибки.

12.1.5 Режимы транзакций

- автофиксация (autocommit);
- неявная транзакция (implicit transaction);
- явная транзакция (explicit transaction).

12.1.6 Режим автоматической фиксации

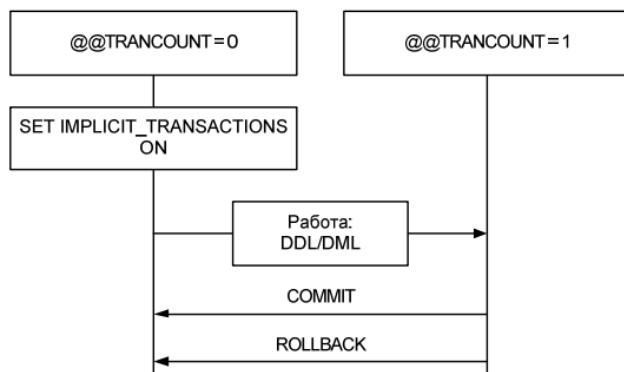
В режиме автоматической фиксации модификация данных и инструкции DDL языка T-SQL выполняются как транзакция, которая будет автоматически зафиксирована, если инструкция выполнена без ошибок, в противном случае будет выполнен откат транзакции. Режим автоматической фиксации является режимом управления транзакциями по умолчанию.

В режиме автоматической фиксации не требуется никаких других команд управления транзакциями, таких как BEGIN TRAN, ROLLBACK TRAN или COMMIT TRAN.

12.1.7 Режим неявных транзакций

В режиме неявных транзакций, когда выполняется одна или более инструкция DML или DDL либо инструкция SELECT, SQL Server запускает транзакцию, увеличивает @@TRANCOUNT, но не выполняет автоматической фиксации или отката инструкции.

```
1 SET IMPLICIT_TRANSACTIONS ON;
```



К преимуществам использования неявных транзакций относятся следующие:

- можно откатить неявную транзакцию после того, как была выполнена команда;
- поскольку инструкцию COMMIT нужно запускать автоматически, у вас есть возможность обнаружить ошибки после окончания выполнения команды.

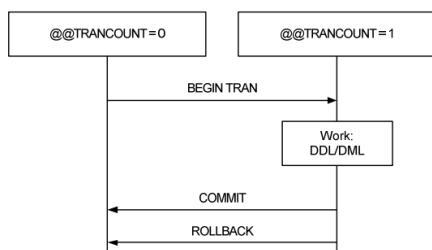
К недостаткам неявных транзакций относятся следующие.

- Любые блокировки, вызванные командой, сохраняются до завершения транзакции. Поэтому вы можете заблокировать других пользователей при выполнении их задач.

- Поскольку это нестандартный метод использования SQL Server, вы должны постоянно помнить о необходимости его установки для вашего сеанса.
- Режим неявных транзакций плохо работает с явными транзакциями, поскольку приводит к неожиданному увеличению значения @@TRANCOUNT до 2.
- Если вы забудете зафиксировать неявную транзакцию, то можете оставить блокировки открытыми. Помните, что неявные транзакции могут входить в пакеты.

12.1.8 Режим явных транзакций

Явная транзакция выполняется, если для запуска транзакции явно указана команда BEGIN TRANSACTION или BEGIN TRAN.



12.1.9 Вложенные транзакции

Когда явные транзакции вложены, т. е. расположены одна в другой, они называются вложенными транзакциями. Поведение инструкций COMMIT и ROLLBACK изменяется для вложенных транзакций.

СОВЕТ

Подготовка к экзамену

Внутренняя инструкция COMMIT не имеет фактического влияния на транзакцию, только увеличивает @@TRANCOUNT на 1. Только самая внешняя инструкция COMMIT, та, которая выполняется при @@TRANCOUNT = 1, действительно фиксирует транзакцию.

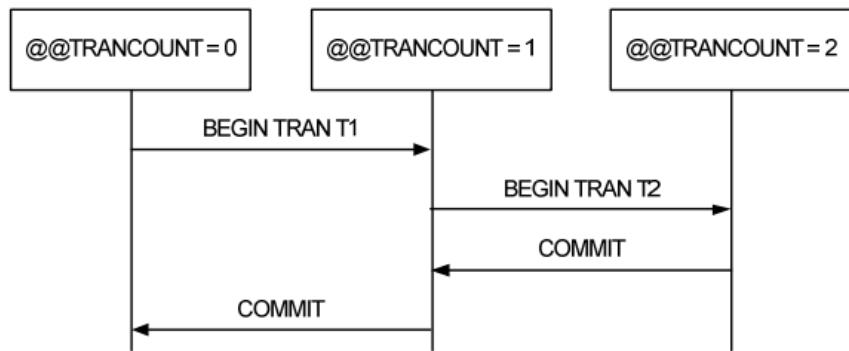


Рис. 12.1: Финальная самая внешняя инструкция COMMIT во вложенной транзакции

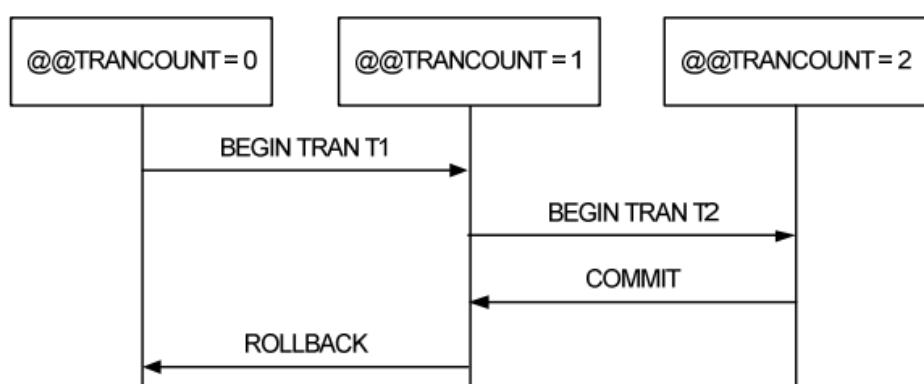


Рис. 12.2: Конечная инструкция ROLLBACK, выполняющая откат всей транзакции

СОВЕТ

Подготовка к экзамену

Обратите внимание, что значение, на котором запущена команда ROLLBACK. Транзакция может содержать только одну команду ROLLBACK, которая выполнит откат всей транзакции и сбросит счетчик @@TRANCOUNT в 0.

12.1.10 Разметка транзакций

Именованные транзакции используются для помещения метки в журнал транзакций, чтобы указать точку, в которую могут быть восстановлены одна или более баз данных.

```

1 USE TSQL2012;
2 BEGIN TRAN Tran1 WITH MARK;
3 ...
4 COMMIT TRAN;
    
```

```
5   ...
6
7   ...
8 RESTORE DATABASE TSQ2012 FROM DISK = 'C:\SQLBackups\TSQL2012.bak',
9 WITH NORECOVERY;
10
11 RESTORE LOG TSQL2012 FROM DISK = 'C:\SQLBackups\TSQL2012.trn',
12 WITH STOPATMARK = 'Tran1';
```

При использовании предложения WITH MARK необходимо помнить следующее.

- Вы должны использовать имя транзакции с ключевым словом WITH STOPATMARK.
- Вы можете поместить описание после предложения WITH MARK, но SQL Server будет его игнорировать.
- Вы можете выполнить восстановление на момент непосредственно перед транзакцией с помощью предложения STOPBEFOREMARK.
- Можно восстановить базу данных из копии с использованием ключевых слов WITH STOPATMARK или STOPBEFOREMARK.
- Вы можете добавить параметр RECOVERY в списке WITH, но это не будет иметь никакого эффекта.

12.1.11 Дополнительные параметры транзакции

- **Точки сохранения.** Это точки внутри транзакций, которые можно использовать для отката выборочной части работы. Можно определить точку сохранения с помощью команды SAVE TRANSACTION.
- **Межбазовые транзакции.** Транзакция может охватывать одну или более баз данных на одном экземпляре SQL Server без каких-либо дополнительных действий на стороне пользователя.

- **Распределенные транзакции.** Транзакция может распространяться на несколько серверов с помощью связанного сервера. Такая транзакция называется распределенной (в противоположность локальной) транзакцией.

КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Сколько команд `ROLLBACK` должно быть выполнено во вложенной транзакции для того, чтобы выполнить ее откат?
2. Сколько инструкций `COMMIT` должно быть выполнено во вложенной транзакции, чтобы гарантировать фиксацию всей транзакции полностью?

Ответы на контрольные вопросы

1. Только одна команда `ROLLBACK`. Команда `ROLLBACK` всегда делает откат всей транзакции, независимо от того, сколько уровней вложенности имеет транзакция.
2. Одна инструкция `COMMIT` на каждый уровень вложенной транзакции. Только последняя инструкция `COMMIT` действительно фиксирует всю транзакцию.

12.1.12 Основные блокировки

Чтобы сохранить изоляцию транзакций, SQL Server реализует набор протоколов блокирования. На базовом уровне существуют два основных режима блокировок:

- совмещаемые блокировки (shared locks) используются для сеансов, выполняющих чтение данных, т. е. для операций считывания;
- монопольные блокировки (exclusive locks) используются для изменений данных, т. е. для операций записи.

КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Могут ли модули чтения (совмещаемые блокировки) блокировать модули чтения?
2. Могут ли модули чтения блокировать модули записи (монопольные блокировки)?

Ответы на контрольные вопросы

1. Нет, поскольку совмещаемые блокировки совместимы с другими совмещаемыми блокировками.
2. Да, даже кратковременный запрос, поскольку любой запрос монопольной блокировки должен ждать, пока совмещаемая блокировка не будет освобождена.

КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Если две транзакции никогда не блокируют друг друга, может ли возникнуть между ними взаимоблокировка?
2. Может ли инструкция SELECT участвовать во взаимоблокировке?

Ответы на контрольные вопросы

1. Нет. Чтобы возникла взаимоблокировка, каждая транзакция должна иметь заблокированный ресурс, который нужен другой транзакции, что приводит к взаимному блокированию.
2. Да. Если инструкция SELECT блокирует некоторый ресурс, не позволяя завершиться другой транзакции, и сама инструкция SELECT не может завершиться, т. к. она заблокирована какой-то транзакцией, возникает цикл взаимной блокировки.

12.1.13 Уровни изоляции транзакций

К наиболее часто используемым уровням изоляции относятся следующие.

- **READ COMMITTED.** Это уровень изоляции по умолчанию. Все модули чтения в данном сеансе будут только выполнять чтение изменений данных, которые были зафиксированы. Поэтому все инструкции SELECT будут пытаться получить совмещаемые блокировки, и любые базовые ресурсы данных, которые изменяются в разных сеансах и, следовательно, имеют монопольные блокировки, будут блокировать сеанс READ COMMITTED.
- **READ UNCOMMITTED.** Этот уровень изоляции позволяет модулям чтения читать незафиксированные данные. Эта настройка удаляет совмещаемые блокировки, полученные инструкциями SELECT, так что модули чтения больше уже не блокируются модулями записи. Но результаты инструкции SELECT могут считывать незафиксированные данные, которые были изменены в течение транзакции и позже откатывались к их первоначальному состоянию. Это называется "грязным" чтением данных.
- **READ COMMITTED SNAPSHOT.** Фактически это не новый уровень изоляции; это дополнительный способ использования уровня изоляции по умолчанию READ COMMITTED, уровень изоляции по умолчанию в базе данных Windows Azure SQL.

Следующие уровни изоляции обеспечивают более строгий контроль над

тем, какие данные могут быть прочитаны между транзакциями. Поскольку они могут приводить к еще большему блокированию или большим затратам, они не используются так часто, как более слабые уровни изоляции.

- **REPEATABLE READ.** Этот уровень изоляции, также устанавливаемый на уровне сеанса, гарантирует, что данные, считанные в транзакции, могут быть позднее снова прочитаны в транзакции. Не допускаются операции обновления и удаления уже выбранных строк. В результате совмещаемые блокировки удерживаются до конца транзакции. Однако транзакция может видеть новые строки, добавленные после первой операции чтения; это называется фантомным чтением.
- **SNAPSHOT.** Этот уровень изоляции также использует управление версиями строк в базе данных tempdb (как это делает RCSI). Он разрешен как постоянное свойство базы данных и поэтому устанавливается на отдельную транзакцию. Транзакция, использующая уровень изоляции SNAPSHOT, сможет повторить любую операцию чтения, и при этом она не будет видеть никаких фантомных чтений. Новые строки могут быть добавлены в таблицу, но транзакция не будет их видеть. Поскольку она использует управление версиями строк, уровень изоляции SNAPSHOT не требует совмещаемых блокировок на базовых данных.
- **SERIALIZABLE.** Этот уровень изоляции является наиболее строгим и устанавливается на сеанс. На этом уровне все операции чтения являются повторяемыми, и новые строки не разрешены в базовых таблицах, что может удовлетворять условиям инструкций SELECT в транзакции.

СОВЕТ

Подготовка к экзамену

Уровни изоляции устанавливаются на сеанс. Если иной уровень изоляции не установлен на сеанс, все транзакции будут выполняться с использованием уровня изоляции по умолчанию, READ COMMITTED, для локальных экземпляров SQL Server — тоже с уровнем READ COMMITTED. В базе данных Windows Azure SQL уровень изоляции по умолчанию — READ COMMITTED SNAPSHOT.

КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Если ваш сеанс использует уровень изоляции READ COMMITTED, может ли один из ваших запросов выполнять чтение незафиксированных данных?
2. Существует ли способ предотвратить блокирование модулей записи модулями чтения и при этом гарантировать, что модули чтения видят только зафиксированные данные?

Ответы на контрольные вопросы

1. Да, если запрос использует табличные подсказки WITH (NOLOCK) или WITH (READUNCOMMITTED). Значение сеанса для уровня изоляции не изменяется, меняются только характеристики чтения таблицы.
2. Да, это задача параметра в уровне изоляции READ COMMITTED. Модули чтения видят более ранние версии изменений данных для текущих транзакций, но не незафиксированные в данный момент данные.

Резюме занятия

- Все изменения данных в SQL Server происходят в контексте транзакции. Выполнение команды ROLLBACK на любом уровне вложенности транзакции немедленно откатывает всю транзакцию полностью.
- Каждая инструкция COMMIT уменьшает значение @@TRANCOUNT на 1, и только самая внешняя инструкция COMMIT фиксирует всю вложенную транзакцию.
- SQL Server использует блокирование для обеспечения изоляции транзакций.
- Взаимоблокировка может возникнуть между двумя или более сеансами, если каждый сеанс получил несовместимые блокировки, которые нужны другому сеансу для завершения его инструкции. Когда SQL Server видит взаимоблокировку, он выбирает один из сеансов и прерывает пакет.
- SQL Server обеспечивает ACID-свойство, называемое уровнем изоляции, с разной степенью строгости.
- Уровень изоляции READ COMMITTED — это уровень изоляции по

умолчанию для локального SQL Server.

- Параметр изоляции READ COMMITTED SNAPSHOT (RCSI) уровня изоляции по умолчанию позволяет запросам на чтение получать доступ к ранее зафиксированным версиям монопольно заблокированных данных. Это может значительно снизить блокирование и взаимоблокирование. RCSI — это уровень изоляции по умолчанию в базе данных Windows Azure SQL.
- Уровень изоляции READ UNCOMMITTED разрешает сеансу считывать незафиксированные данные, известные как "грязные чтения".

Закрепление материала

1. Какие из следующих инструкций T-SQL выполняются автоматически в контексте транзакции? (Выберите все подходящие варианты.)
 - А. Команда ALTER TABLE.
 - В. Команда PRINT.
 - С. Команда UPDATE.
 - Д. Команда SET.
2. Как команды COMMIT и ROLLBACK работают с вложенными транзакциями в языке T-SQL? (Выберите все подходящие варианты.)
 - А. Одна команда COMMIT фиксирует всю вложенную транзакцию.
 - В. Одна команда ROLLBACK выполняет откат всей вложенной транзакции.
 - С. Одна команда COMMIT фиксирует только один уровень вложенной транзакции.
 - Д. Одна команда ROLLBACK выполняет откат только одного уровня вложенной транзакции.
3. Какая из следующих стратегий может помочь снизить блокирование и взаимоблокирование, уменьшив совмещаемые блокировки? (Выберите все подходящие варианты.)

- А. Добавьте в запросы табличную подсказку READUNCOMMITTED.
- Б. Используйте параметр READ COMMITTED SNAPSHOT.
- С. Используйте уровень изоляции REPEATABLE READ.
- Д. Используйте уровень изоляции SNAPSHOT.

Ответы

Занятие 1

Закрепление материала

1. Правильные ответы: А, С и D.
 - A. **Правильно:** команда ALTER TABLE — это команда языка DDL, которая изменяет метаданные и всегда выполняется как транзакция.
 - B. **Неправильно:** команда PRINT не изменяет данные и поэтому не выполняется отдельно в транзакции.
 - C. **Правильно:** инструкция UPDATE изменяет данные и выполняется как транзакция.
 - D. **Правильно:** инструкция SET влияет только на параметры сеанса и не изменяет данные, поэтому она не выполняется как транзакция.
2. Правильный ответ: В.
 - A. **Неправильно:** одна операция COMMIT фиксирует только самый внутренний уровень транзакции и не будет фиксировать всю вложенную транзакцию.
 - B. **Правильно:** одна операция ROLLBACK выполняет откат всей внешней транзакции во вложенной транзакции.
 - C. **Неправильно:** одна операция COMMIT фиксирует только самый внешний уровень вложенной транзакции.
 - D. **Неправильно:** одна операция ROLLBACK не выполняет откат только одного уровня транзакции; напротив, она осуществляет откат всей транзакции.
3. Правильные ответы: А, В и D.
 - A. **Правильно:** добавление табличной подсказки READUNCOMMITTED не вызывает совмещаемых блокировок, которые могут использоваться инструкцией.
 - B. **Правильно:** параметр READ COMMITTED SNAPSHOT выполняет чтение зафиксированных данных из версий, а не с помощью запроса совмещаемых блокировок.

- C. **Неправильно:** уровень изоляции REPEATABLE READ в действительности удерживает совмещаемые блокировки до конца транзакции и, таким образом, может увеличить блокирование и взаимоблокирование.
- D. **Правильно:** уровень изоляции SNAPSHOT также снижает совмещаемые блокировки посредством чтения зафиксированных данных из зафиксированных версий, а не с помощью совмещаемых блокировок, поэтому он также может уменьшить блокирование и взаимоблокирование.

12.2 Реализация обработки ошибок

12.2.1 Анализ сообщений об ошибке

Далее приведен пример сообщения об ошибке от SQL Server 2012.

```
Msg 547, Level 16, State 0, Line 11
The INSERT statement conflicted with the FOREIGN KEY constraint
"FK_Products_Categories".
The conflict occurred in database "TSQL2012", table "Production.Categories",
column 'categoryid'.
```

Сообщения об ошибке в SQL Server состоит из четырех частей

- **Номер ошибки** (error number) — это целочисленное значение. Нумеруются от 1 до 49 999. Пользовательские сообщения об ошибках нумеруются с 50 001 и выше.
- **Уровень серьезности**. В SQL Server определено 26 уровней серьезности с номерами от 0 до 25. По общему правилу, ошибки с уровнем серьезности от 16 и выше автоматически записываются в журнал SQL Server и в журнал приложений Windows. Ошибки с уровнем серьезности от 19 до 25 могут быть указаны только членами предопределенной роли сервера sysadmin. Ошибки с уровнем серьезности от 20 до 25 считаются фатальными (неустранимыми) и приводят к разрыву соединений и откату всех открытых транзакций. Ошибки с уровнем серьезности 0 до 10 являются информационными.
- **Состояние (state)** — целое число с максимальным значением 127, используемое компанией Microsoft для внутренних целей.
- **Сообщение об ошибке** (error message) может иметь длину до 255 символов в кодировке Unicode. Сообщения об ошибках SQL Server перечислены в представлении каталога sys.messages. Можно добавлять пользовательские сообщения об ошибках с помощью процедуры sp_addmessage.

12.2.2 Команда THROW

Команда THROW во многом ведет себя так же, как команда RAISERROR, за некоторыми важными исключениями. Базовый синтаксис команды THROW выглядит следующим образом:

```
1  THROW [ { error_number | @local_variable },
2  { message | @local_variable },
3  { state | @local_variable }
4  ] [; ]
```

Команда THROW имеет множество таких же компонентов, что и команда RAISERROR, но со следующими значительными отличиями:

- команда THROW может использоваться без параметров, но только в блоке CATCH конструкции TRY/CATCH;
- если имеются параметры, то номер ошибки, сообщение и состояние являются обязательными;
- номер ошибки error_number не требует соответствующего сообщения, определенного в представлении каталога sys.messages;
- параметр сообщения не допускает форматирование, но можно использовать функцию FORMATMESSAGE() с переменной для получения того же результата;
- параметр состояния (state) должен быть целым числом в диапазоне от 0 до 255;
- любой параметр может быть переменной;
- параметр серьезности ошибки не используется, уровень серьезности ошибки всегда устанавливается равным 16;
- команда THROW всегда прерывает выполнение пакета, за исключением случаев, когда она используется в блоке TRY.

В качестве примера можно вызвать простую инструкцию THROW следующим образом:

COBET**Подготовка к экзамену**

Инструкция, выполняющаяся перед инструкцией THROW, должна завершаться точкой с запятой (;). Считается наиболее правильным завершать все инструкции T-SQL точкой с запятой.

```
1   THROW 50000, 'Error in usp_InsertCategories stored procedure', 0;
```

Функции TRY_CONVERT и TRY_PARSE

Функция TRY_CONVERT пытается привести значение к выходному типу данных и в случае успеха возвращает это значение, в противном случае возвращается значение NULL.

```
1   SELECT TRY_CONVERT(DATETIME, '1752-12-31');
2   SELECT TRY_CONVERT(DATETIME, '1753-01-01');
```

Функция TRY_PARSE позволяет принимать входную строку, содержащую данные с неопределенным типом данных и преобразовывать их в конкретный тип данных при возможности, или возвращать NULL в противном случае. В следующем примере выполняет синтаксический анализ двух строк.

```
1   SELECT TRY_PARSE('1' AS INTEGER);
2   SELECT TRY_PARSE('B' AS INTEGER);
```

КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Как можно добавлять пользовательские сообщения об ошибках?
2. Для чего используется уровень серьезности, равный 0?

Ответы на контрольные вопросы

1. Чтобы добавить собственные пользовательские сообщения об ошибках, вы можете применять системную хранимую процедуру sp_addmessage.
2. При использовании команды RAISERROR с уровнем серьезности 0 отправляется только информационное сообщение. Если вы добавите параметр WITH NOWAIT, сообщение будет отправляться без ожидания во внешнем буфере.

12.2.3 Неструктурированная обработка ошибок с помощью функции @@ERROR

Неструктурированная обработка ошибок заключается в проверке отдельных инструкций на их состояние ошибки непосредственно после их выполнения. Для этого необходимо запросить системную функцию @@ERROR. Основная проблема неструктурной обработки ошибок — необходимость проверять @@ERROR после каждой модификации данных или инструкции DML. Поскольку центральное местоположение для обработки ошибок не предоставляется, обработка ошибок должна выполняться в пользовательском коде.

12.2.4 Использование параметра XACT_ABORT с транзакциями

Поставив в начало пакета SET XACT_ABORT ON, можно вызвать сбой всего этого пакета в случае возникновения ошибки. XACT_ABORT устанавливается на отдельный сеанс.

12.2.5 Структурированная обработка ошибок с помощью конструкции TRY/CATCH

Далее приведены некоторые правила использования конструкции TRY/CATCH

- Ошибки с уровнем серьезности больше 10 и меньше 20 в блоке TRY приводят к передаче управления блоку CATCH.
- Ошибки с уровнем серьезности больше 10 и меньше 20 в блоке TRY приводят к передаче управления блоку CATCH.
- Ошибки компиляции и некоторые ошибки выполнения программы, задействующие компиляцию уровня инструкции, прерывают выполнение пакета немедленно и не передают управление CATCH.

- Если ошибка произошла в блоке CATCH, транзакция прерывается и ошибка возвращается вызывающему приложению, если блок CATCH не вложен в блок TRY.
- В блоке CATCH можно выполнить фиксацию или откат текущей транзакции, если транзакция не может быть зафиксирована и ее необходимо откатить. Для проверки состояния транзакции можно запросить функцию XACT_STATE.
- Блок TRY/CATCH не перехватывает ошибки, приводящие к прерыванию соединения, такие как неустранимая ошибка или выполнение роля sysadmin команды KILL.
- Также невозможно перехватить ошибки, которые возникают из-за ошибок компиляции, синтаксических ошибок или несуществующих объектов. Поэтому вы не можете использовать конструкцию TRY/CATCH для проверки существования объекта.
- Блоки TRY/CATCH могут быть вложенными; другими словами, можно поместить внутренний блок TRY/CATCH во внешний блок TRY. Ошибка внутри вложенного блока TRY передает выполнение соответствующему вложенному блоку CATCH.

```

1 BEGIN CATCH
2     SELECT ERROR_NUMBER() AS errornumber
3     , ERROR_MESSAGE() AS errormessage
4     , ERROR_LINE() AS errorline
5     , ERROR_SEVERITY() AS errorseverity
6     , ERROR_STATE() AS errorstate;
7 END CATCH;
```

Резюме занятия

- SQL Server 2012 использует команды RAISERROR и THROW для генерации ошибок.

При обработке ошибок в блоке CATCH номера ошибок, которые могут произойти, являются довольно большими, поэтому трудно их все предусмотреть. Также возможны специализированные типы транзакций или процедур. Некоторые разработчики T-SQL предпочитают просто возвращать значения функций ошибок, как в предыдущей инструкции SELECT. Это может быть наиболее полезным для утилитных хранимых процедур. В других случаях некоторые разработчики T-SQL используют хранимую процедуру, которая может быть вызвана из блока CATCH, и это обеспечит общую реакцию на определенные наиболее часто встречающиеся ошибки.

СОВЕТ**Подготовка к экзамену**

Вы должны проследить, чтобы инструкция THROW с параметрами или без них была последней инструкцией, которая должна быть выполнена в блоке CATCH, поскольку она прерывает выполнение пакета, и оставшиеся команды в блоке CATCH не выполняются.

КОНТРОЛЬНЫЕ ВОПРОСЫ

1. В чем заключаются главные преимущества использования блока TRY/CATCH по сравнению с традиционным захватом ошибок функцией @@ERROR?
2. Может ли блок TRY/CATCH охватывать пакеты?

Ответы на контрольные вопросы

1. Главное преимущество заключается в том, что у вас есть единое пространство в вашем коде, в которое будут захватываться ошибки, так что вам нужно всего лишь поместить обработку ошибок в одно место.
2. Нет, необходимо иметь один набор блоков TRY/CATCH для каждого пакета кода.

- Можно запросить системную функцию @@ERROR для определения, произошла ли ошибка и какой у нее номер.
- Можно использовать команду SET XACT_ABORT ON, чтобы вызвать сбой транзакции и прерывание пакета, когда возникнет ошибка.
- Неструктурированная обработка ошибок не предоставляет единого места в коде для обработки ошибок.
- Блок TRY/CATCH предоставляет каждому блоку кода T-SQL блок CATCH, в котором выполняется обработка ошибок.
- Команда THROW может быть использована для повторной выдачи ошибок.
- Существует полный набор функций обработки ошибок для сбора ин-

формации об ошибках.

Закрепление материала

1. В чем заключается преимущество использования команды `THROW` в блоке `CATCH`?
 - A. Команда `THROW` в блоке `CATCH` не требует параметров и поэтому более проста в написании.
 - B. Команда `THROW` повторно вызывает исходную ошибку, так что эта ошибка может быть обработана.
 - C. Команда `THROW` автоматически использует уровень серьезности 16.
 - D. Инструкция перед командой `THROW` требует точки с запятой.
2. Какие из перечисленных функций могут использоваться в блоке `CATCH` для возвращения информации об ошибке? (Выберите все подходящие варианты.)
 - A. `@@ERROR`.
 - B. `ERROR_NUMBER()`.
 - C. `ERROR_MESSAGE()`.
 - D. `XACT_STATE()`.
3. Как инструкция `SET XACT_ABORT ON` влияет на транзакцию?
 - A. Если возникает ошибка T-SQL с уровнем серьезности более 16, транзакция будет прервана.
 - B. Если возникает ошибка T-SQL с уровнем серьезности более 10, транзакция будет прервана.
 - C. Если возникает ошибка T-SQL с уровнем серьезности более 16, некоторые инструкции транзакции тем не менее могут быть выполнены.
 - D. Если возникает ошибка T-SQL с уровнем серьезности более 10, некоторые инструкции транзакции тем не менее могут быть выполнены.

Ответы

1. Правильный ответ: В.
 - A. **Неправильно:** действительно, команда `THROW` не принимает параметры в блоке `CATCH`, но это не обязательно является преимуществом.
 - B. **Правильно:** инструкция `THROW` в блоке `CATCH` может повторно вызвать ошибку и таким образом позволить создать сообщение об ошибке в блоке `TRY` без необходимости сохранения какой-либо предварительной информации. Это позволяет выполнять обработку ошибок в блоке `CATCH`.
 - C. **Неправильно:** инструкция `THROW` всегда дает уровень серьезности 16, но это не обязательно является преимуществом. Команда `RAISERROR` является более гибкой, предоставляя диапазон уровней серьезности.
 - D. **Неправильно:** требование наличия точки с запятой в предыдущей инструкции T-SQL — возможно, правильное требование написания кода, но это не является преимуществом, предоставляемым командой `THROW`.
2. Правильные ответы: А, В, С и Д.
 - A. **Правильно:** значение `@@ERROR` изменяется с каждой успешной командой, поэтому если доступ к нему может быть получен в самой первой инструкции блока `CATCH`, можно получить исходное сообщение об ошибке.
 - B. **Правильно:** `ERROR_NUMBER()` возвращает номер ошибки для исходной ошибки, которая привела к передаче управления блоку `CATCH`.
 - C. **Правильно:** `ERROR_MESSAGE()` возвращает текст исходной ошибки.
 - D. **Правильно:** `XACT_STATE()` сообщает состояние транзакции в блоке `CATCH`, в частности, является ли транзакция фиксируемой.
3. Правильный ответ: В.
 - A. **Неправильно:** ошибка T-SQL с уровнем серьезности более 16 не вызывает прерывания всей транзакции.
 - B. **Правильно:** ошибка T-SQL с уровнем серьезности более 10 вызывает прерывание транзакции.

- C. **Неправильно:** когда транзакция прервана параметром `XACT_ABORT`, никакие другие инструкции в транзакции не будут выполнены.
- D. **Неправильно:** когда транзакция прервана параметром `XACT_ABORT`, никакие другие инструкции в транзакции не будут выполнены.

12.3 Использование динамического SQL

Динамический SQL является полезной возможностью, т. к. T-SQL не разрешит прямую замену многих частей команд переменными, включая следующие:

- имя базы данных в инструкции USE;
- имена таблиц в предложении FROM;
- имена столбцов в предложениях SELECT, WHERE, GROUP BY и HAVING, а также в предложении ORDER BY;
- содержимое списков, например, в предложениях IN и PIVOT.
- Генерация кода для автоматизации административных задач.
- Выполнение итераций во всех базах данных на сервере, во всех типах объектов в базе данных и в метаданных объектов, таких как имена столбцов или индексы.
- Построение хранимых процедур с множеством необязательных параметров, составляющих результирующие запросы, на основании которых параметры получают значения.

12.3.1 Инструкция EXECUTE

Простейший способ, предоставляемый SQL Server для выполнения динамического SQL, — это инструкция EXECUTE, которую можно записывать как EXECUTE или сокращенно EXEC.

Далее перечислены особенности использования команды EXEC, которые следует знать.

- Входная строка должна быть одним пакетом T-SQL. Стока может содержать множество команд T-SQL, но она не может содержать разделители GO.

- Можно использовать строковые литералы, строковые переменные или их объединение.

```

1  DECLARE @SQLString AS NVARCHAR(MAX)
2  , @tablename AS NVARCHAR(261) = '[Production].[Products]';
3  SET @SQLString = 'SELECT COUNT(*) AS TableRowCount FROM '
4  EXEC(@SQLString + @tablename);

```

Контрольные вопросы

1. Можно ли сгенерировать и выполнить динамический SQL в другой базе данных по отношению к той, в которой находится код?
2. На какие объекты нельзя ссылаться в T-SQL с помощью переменных?

Ответы на контрольные вопросы

1. Да, поскольку команда `USE <database>` может быть вставлена в динамический пакет SQL.
2. К объектам, для которых нельзя использовать переменные в командах T-SQL, относятся имя базы данных в инструкции `USE`, имя таблицы в предложении `FROM`, имена столбцов в предложениях `SELECT` и `WHERE` и списки литеральных значений в функциях `IN()` и `PIVOT()`.

Контрольные вопросы

1. Как может определить хакер, что существует возможность внедрения SQL-кода?
2. Где вставляется внедренный код?

Ответы на контрольные вопросы

1. Вставив одиночную кавычку и проверив сообщения об ошибке.
2. Между начальной одиночной кавычкой, которая прерывает строку входных данных, и конечным знаком комментария, который запрещает внутреннюю прерывающую одиночную кавычку.

12.3.2 Использование хранимой процедуры `sp_executesql`

Поддерживаются выходные параметры. Благодаря параметрам, системная хранимая процедура `sp_executesql` является более безопасной и может помочь предотвратить некоторые виды внедрения SQL-кода. Параметры `sp_executesql` не могут быть использованы для замещения необходимых строковых литералов, таких как имена таблиц и столбцов.

```
1 sp_executesql [ @statement = ] statement  
2 [ {, [ @params = ] N'@parameter_name data_type [ OUT | OUTPUT ][,...n ]', }  
3 {, [ @param1 = ] 'value1' [ ,...n ] }]
```

Входной параметр @statement имеет тип данных NVARCHAR(MAX). Вам нужно представить инструкцию в виде Unicode-строки в параметр @statement и встроить в эту инструкцию параметры, которые должны быть заменены в финальной строке. Вы должны перечислить имена этих параметров вместе с их типами данных в строке @params, а затем поместить значения в списки @param1, @param2 и т. д.

СОВЕТ

Подготовка к экзамену

Способность выполнять параметризацию означает, что процедура sp_executesql избегает простых объединений, подобных используемым инструкцией EXEC. В результате она может быть применена для предотвращения внедрения SQL-кода.

Хранимая процедура sp_executesql иногда обеспечивает лучшую производительность запроса, чем команда EXEC, т. к. параметризация способствует повторенному использованию кэшированных планов выполнения. Поскольку хранимая процедура sp_executesql принуждает к параметризации, часто строка запроса остается той же самой, а изменяются только значения параметров.

КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Как можно передать информацию из хранимой процедуры sp_executesql вызывающей стороне?
2. Как хранимая процедура sp_executesql может остановить внедрение SQL-кода?

Ответы на контрольные вопросы

1. Необходимо использовать один или более параметров OUTPUT. Также можно сохранить данные во временной или постоянной таблице, но использование параметра OUTPUT является наиболее очевидным способом.
2. Можно использовать хранимую процедуру sp_executesql для параметризации входных данных пользователя, что может предотвратить выполнение любого внедренного кода.

Резюме занятия

- Динамический SQL можно использовать для генерации и выполнения кода T-SQL в случаях, когда инструкции T-SQL должны строиться во время выполнения.
- Внедрение SQL-кода означает потенциальную возможность для приложений принимать входные данные, которые внедряют нежелательный код, выполняемый динамическим SQL.
- Хранимая процедура sp_executesql может использоваться, чтобы помочь предотвратить внедрение SQL-кода, путем параметризации соответствующих частей динамического SQL.

Закрепление материала

1. Какие из следующих способов можно использовать для внедрения нежелательного кода в динамический SQL, когда пользовательские входные данные объединены с допустимыми командами SQL?
 - A. Вставьте строку комментария из двух тире, потом вредоносный код и затем одиночную кавычку.
 - B. Вставьте одиночную кавычку, потом вредоносный код и затем строку комментария из двух тире.
 - C. Вставьте вредоносный код перед одиночной кавычкой и строку комментария из двух тире.
2. В чем заключаются преимущества хранимой процедуры sp_executesql над командой EXECUTE ()? (Выберите все подходящие варианты.)
 - A. Хранимая процедура sp_executesql может выполнять параметризацию аргументов поиска и помочь предотвратить внедрение SQL-кода.
 - B. Хранимая процедура sp_executesql использует строки в формате Unicode.
 - C. Хранимая процедура sp_executesql может возвращать данные посредством выходных параметров.
3. Какие из следующих высказываний справедливы по отношению к инструкции SET QUOTED_IDENTIFIER? (Выберите все подходящие варианты.)
 - A. Когда параметр QUOTED_IDENTIFIER установлен в ON, инструкция позволяет использовать двойные кавычки для разделения идентификаторов T-SQL, таких как имена таблиц и столбцов.
 - B. Когда параметр QUOTED_IDENTIFIER установлен в OFF, инструкция позволяет использовать двойные кавычки для разделения идентификаторов T-SQL, таких как имена таблиц и столбцов.

- C. Когда параметр QUOTED_IDENTIFIER установлен в ON, инструкция позволяет использовать двойные кавычки для разделения строк.
- D. Когда параметр QUOTED_IDENTIFIER установлен в OFF, инструкция позволяет использовать двойные кавычки для разделения строк.

Ответы

.....

1. Правильный ответ: В.
 - A. **Неправильно:** строка комментария должна стоять последней, а внедренная одиночная кавычка — первой.
 - B. **Правильно:** начальная одиночная кавычка прерывает входную строку, а конечный знак комментария уничтожает эффект прерывающей одиночной кавычки. Поэтому вредоносный код может быть вставлен между ними.
 - C. **Неправильно:** вредоносный код должен стоять после первой одиночной кавычки и перед конечным знаком комментария.
2. Правильные ответы: А и С.
 - A. **Правильно:** параметризация — главное преимущество хранимой процедуры `sp_executesql` над инструкцией `EXEC()`, поскольку она гарантирует, что любой внедренный код будет виден только как значение строкового параметра, а не как исполняемый код.
 - B. **Неправильно:** хотя хранимая процедура `sp_executesql` требует строк Unicode в качестве параметров, это не обязательно является преимуществом. Команда `EXECUTE` принимает и формат Unicode, и отличные от него форматы, и таким образом может считаться более гибкой.
 - C. **Правильно:** использование выходных параметров разрешает многие ограничения команды `EXECUTE`. Команда `EXECUTE` не может напрямую возвращать информацию вызываемому сеансу.
3. Правильные ответы: А и D.
 - A. **Правильно:** если параметр `QUOTED_IDENTIFIER` установлен в `ON`, можно использовать двойные кавычки для разделения идентификаторов T-SQL, таких как имена таблиц и столбцов.
 - B. **Неправильно:** если параметр `QUOTED_IDENTIFIER` установлен в `OFF`, нельзя использовать двойные кавычки для разделения идентификаторов T-SQL, таких как имена таблиц и столбцов.
 - C. **Неправильно:** если параметр `QUOTED_IDENTIFIER` установлен в `ON`, нельзя использовать двойные кавычки для разделения строк.
 - D. **Правильно:** если параметр `QUOTED_IDENTIFIER` установлен в `OFF`, можно использовать двойные кавычки для разделения строк.

Упражнения

Упражнение 1. Реализация обработки ошибок

Как разработчика баз данных на ключевом проекте в компании, вас попросили выполнить рефакторинг набора хранимых процедур на промышленном сервере баз данных. Вы обнаружили, что в хранимых процедурах практически отсутствует обработка ошибок, а когда она выполняется, она является ситуативной и неструктурированной. Хранимые процедуры не используют транзакции. Вам необходимо предложить план ваших действий.

1. Когда вы должны рекомендовать использование явных транзакций?
2. Когда вы должны рекомендовать использование другого уровня изоляции?
3. Какой тип обработки ошибок вы должны рекомендовать?
4. Какие планы вы должны включить в рефакторинг динамического SQL?

Упражнение 2. Реализация транзакций

Вас только что назначили в команду на новый проект в качестве разработчика баз данных. Приложение будет использовать хранимые процедуры для выполнения некоторых финансовых операций. Вы решили использовать транзакции T-SQL. Ответьте на следующие вопросы о том, что вы должны порекомендовать в указанных ситуациях.

1. В некоторых транзакциях, обновляющих таблицы, после того как сеанс выполняет чтение определенного значения из другой таблицы, важно, чтобы значение из другой таблицы не изменялось до окончания транзакции. Какой уровень изоляции транзакции подходит для этого?
2. Вы будете использовать сценарии T-SQL для развертывания новых объектов, таких как таблицы, представления или код T-SQL в базе данных. Если возникает любой тип ошибки T-SQL, весь сценарий развертывания должен прекратить выполняться. Как этого достигнуть, не добавляя сложной обработки ошибок?
3. Одна из хранимых процедур будет передавать деньги с одного счета на другой. В течение этого периода передачи никакие данные в обоих счетах не должны быть изменены, вставлены или удалены для диапазона значений, считываемых транзакцией. Какой уровень изоляции транзакции подходит для этого?

Ответы

Упражнение 1. Реализация обработки ошибок

1. Всякий раз, когда происходит более чем одно изменение данных в хранимой процедуре, и важно, чтобы изменения данных обрабатывались как логическая единица работы, необходимо добавлять логику транзакции в хранимую процедуру.
2. Вам необходимо привести уровни изоляции в соответствие требованиям согласованности транзакций. Вы должны изучить существующее приложение и базу данных на наличие элементов блокирования и особенно взаимоблокирования. Если вы найдете взаимоблокировки и установите, что они вызваны не ошибками в кодах T-SQL, можно использовать различные способы снижения уровня изоляции, чтобы сделать взаимоблокировки менее вероятными. Но следует учитывать, что некоторые транзакции могут требовать более высоких уровней изоляции.
3. Вам следует использовать блоки TRY/CATCH в каждой хранимой процедуре, где могут возникать ошибки, и рекомендовать вашей команде сделать такое использование стандартом. Направляя все ошибки в блок CATCH, можно обрабатывать ошибки в одном месте кода.
4. Проверьте хранимые процедуры на использование динамического SQL и где возможно замените вызовы команды EXECUTE хранимой процедурой sp_executesql.

Упражнение 2. Реализация транзакций

1. Чтобы гарантировать, что ни при каком чтении данных в транзакции данные не будут изменяться до конца транзакции, можно использовать уровень изоляции транзакции REPEATABLE READ. Это наименее строгий уровень, который будет удовлетворять требованиям.
2. При развертывании новых объектов базы данных с помощью сценариев T-SQL можно запаковать пакеты в одну транзакцию и использовать настройку SET XACT_ABORT ON сразу после инструкции BEGIN TRANSACTION. Затем, если возникнет какая-либо ошибка T-SQL, вся транзакция будет прервана, и вам не придется добавлять сложную обработку ошибок.
3. Чтобы гарантировать, что для чтения диапазона значений транзакцией никакие строки, для которых выполняется чтение, не были изменены и что никакие строки не смогут быть вставлены либо удалены, можно использовать уровень изоляции SERIALIZABLE. Это наиболее строгий уровень изоляции, который может привести к множеству блокировок, поэтому следует убедиться, что транзакции будут завершаться как можно быстрее.

13 Разработка и реализация процедур T-SQL

13.1 Разработка и реализация хранимых процедур

Хранимые процедуры — это процедуры, которые находятся в базе данных и инкапсулируют код.

13.1.1 Основные сведения о хранимых процедурах

В действительности почти все инструкции T-SQL могут быть включены в хранимую процедуру. Однако надо учесть следующее:

- нельзя использовать команду USE <database name>;
- нельзя использовать инструкцию CREATE с любым из следующих типов объектов: AGGREGATE, RULE, DEFAULT, CREATE, FUNCTION, TRIGGER, PROCEDURE или VIEW;
- можно создать, изменить и удалить таблицу и индекс с помощью инструкций CREATE, ALTER и DROP.

13.1.2 Параметры хранимой процедуры

```
1 CREATE PROC Sales.GetCustomerOrders
2     @custid AS INT,
3     @orderdatefrom AS DATETIME = '19000101',
4     @orderdateto AS DATETIME = '99991231',
5     @numrows AS INT = 0 OUTPUT
6     AS
7     BEGIN
8         SET NOCOUNT ON;
9         SELECT orderid, custid, shipperid, orderdate, requireddate, shippeddate
10        FROM [Sales].[Orders]
11       WHERE custid = @custid
```

```
12    AND orderdate >= @orderdatefrom  
13    AND orderdate < @orderdateto;  
14    SET @numrows = @@ROWCOUNT;  
15    RETURN;  
16 END
```

- Параметры могут быть обязательными или необязательными. При отсутствии инициализации по умолчанию параметр является обязательным. В предыдущем коде @custid — это обязательный параметр.
- Ключевое слово OUTPUT определяет специальный параметр, который возвращает значения вызывающей стороне. Выходные параметры всегда являются необязательными параметрами.
- Команда AS обязательно должна стоять после списка параметров.

13.1.3 Блок BEGIN/END

Можно заключить код в хранимой процедуре в блок BEGIN/END. Хотя это не является обязательным требованием, использование блока BEGIN/END помогает сделать код более понятным.

13.1.4 Инструкция SET NOCOUNT ON

Можно встроить инструкцию NOCOUNT со значением ON внутрь хранимой процедуры, чтобы запретить вывод сообщений, подобный (3 row(s) affected), при каждом выполнении процедуры.

СОВЕТ**Подготовка к экзамену**

Параметр NOCOUNT устанавливается в значение ON или OFF для хранимой процедуры при ее создании. Помещение SET NOCOUNT ON в начало каждой хранимой процедуры предотвращает возвращение этой процедурой сообщения клиенту. Кроме того, SET NOCOUNT ON может повысить производительность часто выполняемых хранимых процедур, поскольку требуется меньше сетевых взаимодействий, если сообщение "row(s) affected" не возвращается клиенту.

13.1.5 Команда RETURN и коды возврата

В одной процедуре можно использовать более одной команды RETURN. Эта команда останавливает выполнение процедуры и возвращает управление обратно вызывающей стороне.

При успешном выполнении статус равен 0, при наличии ошибки статус равен отрицательному числу. Но не следует полагаться на номера ошибок, т. к. они не надежны. Вместо этого следует использовать номера ошибок SQL Server, возвращаемые функцией @@ERROR или функцией ERROR_NUMBER(), вызываемой в блоке CATCH.

СОВЕТ**Подготовка к экзамену**

При вызове хранимой процедуры всегда используйте команду EXEC. Это поможет избежать неожиданных и сбивающих с толку ошибок. Даже если инструкция уже не является первой в пакете, она все равно будет выполнена.

СОВЕТ**Подготовка к экзамену**

Наиболее правильным считается указывать имена параметров при вызове хранимых процедур. Хотя передача параметров по позиции может быть более компактной, она при этом является наиболее ненадежной. Если параметры передаются по имени и порядок следования параметров изменится в хранимой процедуре, вызов этой процедуры все равно будет работать.

13.1.6 Логика ветвления

К инструкциям управления потоком выполнения относятся следующие:

- IF/ELSE;
- WHILE (с BREAK и CONTINUE);
- WAITFOR;
- GOTO;
- RETURN (обычно внутри процедур T-SQL).

13.1.7 Конструкция IF/ELSE

```
1  DECLARE @var1 AS INT, @var2 AS INT;
2  SET @var1 = 1;
3  SET @var2 = 2;
4  IF @var1 = @var2
5    PRINT 'The variables are equal';
6  ELSE
7    PRINT '@var1 equals @var2';
8  GO
```

Если инструкции IF или ELSE используются без блока BEGIN/END, каждая из них обрабатывает только одну инструкцию.

13.1.8 Конструкция WHILE

```
1  SET NOCOUNT ON;
2  DECLARE @count AS INT = 1;
3  WHILE @count <= 10
4    BEGIN
5      PRINT CAST(@count AS @count);
6      SET @count += 1;
7    END;
```

СОВЕТ

Подготовка к экзамену

Хотя блок BEGIN/END является необязательным в цикле WHILE, если у вас есть только одна инструкция, считается наиболее правильным использовать его. Блок BEGIN/END помогает правильно организовать ваш код, делает его более удобным для чтения и будущего изменения. Любой блок инструкций в цикле WHILE, содержащий более одной инструкции, требует включения конструкции BEGIN/END.

Важно отметить, что когда условие WHILE тестирует элементы, которые могут содержать дубликаты, оно захватывает только их общее значение, пропуская все остальные.

13.1.9 Команда WAITFOR

```
1  DECLARE @categoryname AS NVARCHAR(15);
2  SET @categoryname = (SELECT MIN(categoryname) FROM Production.Categories);
3  WHILE @categoryname IS NOT NULL
4  BEGIN
5    PRINT @categoryname;
6    SET @categoryname = (SELECT MIN(categoryname) FROM Production.Categories
7    WHERE categoryname > @categoryname);
8  END;
9  GO
```

13.1.10 Команда WAITFOR

Команда WAITFOR имеет три варианта: WAITFOR DELAY, WAITFOR TIME и WAITFOR RECEIVE (WAITFOR RECEIVE используется только с компонентом Service Broker). Опция WAITFOR DELAY вызывает задержку выполнения на указанный период времени. Например, следующий код останавливает выполнение кода на 20 секунд.

```
1  WAITFOR DELAY '00:00:20';
```

Опция WAITFOR TIME приостанавливает выполнение до указанного времени. Например, следующий код ждет до 23:45.

```
1  WAITFOR TIME '23:46:00';
```

13.1.11 Вызов других хранимых процедур

Если временная таблица создается в одной хранимой процедуре — например, назовем ее Proc1, — эта временная таблица видна всем другим хранимым процедурам, вызываемым из Proc1. Но эта временная таблица не видна процедурам, вызывающим Proc1. Переменные, объявленные в Proc1, и параметры процедуры Proc1 не видны никаким процедурам, вызываемым процедурой Proc1.

КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Назовите два типа параметров для хранимых процедур T-SQL.
2. Может ли хранимая процедура охватывать несколько пакетов T-SQL?

Ответы на контрольные вопросы

1. Хранимая процедура T-SQL может иметь входные и выходные параметры.
2. Нет, хранимая процедура может содержать только один пакет кода T-SQL.

Резюме занятия

- Хранимые процедуры T-SQL — это модули кода T-SQL, которые хранятся в базе данных и могут быть выполнены с помощью команды T-SQL EXECUTE.
- Хранимые процедуры можно использовать для инкапсуляции кода на стороне сервера, что снижает нагрузку на сеть со стороны приложений; для представления приложениям уровня доступа к данным; для выполнения административных задач и техобслуживания.
- Хранимые процедуры можно определить с помощью параметров. Входные параметры отправляются хранимой процедуре для внутреннего использования процедуры. Выходные параметры могут использоваться для возвращения информации вызывающей стороне.
- Внутри хранимой процедуры параметры определяются с помощью такого же синтаксиса, что и переменные T-SQL, и на них можно ссылаться и манипулировать ими внутри процедуры точно так же, как это происходит с переменными.
- Каждая хранимая процедура состоит только из одного пакета кода T-SQL. Хранимые процедуры могут вызывать другие хранимые процедуры.

- Где бы ни выполнялась команда RETURN, выполнение хранимой процедуры заканчивается и управление возвращается вызывающей стороне.
- Хранимые процедуры могут возвращать вызывающей стороне более одного результирующего набора.

Закрепление материала

1. Какие из следующих инструкций T-SQL могут использоваться для создания ветвления в хранимой процедуре? (Выберите все подходящие варианты.)
 - WHILE.
 - BEGIN/END.
 - IF/ELSE.
 - GO.
2. Хранимая процедура вызывает другую хранимую процедуру. Вызываемая хранимая процедура создала временные таблицы, объявила переменные и передала параметры вызываемой хранимой процедуре. Какие данные вызывающей стороны может видеть вызываемая хранимая процедура?
 - Хранимая процедура может видеть переменные, временные таблицы и переданные параметры вызывающей стороны.
 - Хранимая процедура может видеть переменные, но не может видеть временные таблицы и переданные параметры вызывающей стороны.
 - Вызываемая процедура может видеть переданные параметры и временные таблицы, но не может видеть переменные вызывающей стороны.
 - Вызываемая процедура не может видеть никаких объектов, созданных вызывающей процедурой.
3. Как можно использовать выходные параметры в хранимых процедурах T-SQL? (Выберите все подходящие варианты.)
 - С помощью выходного параметра можно передать данные в процедуру, но нельзя получить информацию обратно из нее.
 - С помощью выходного параметра можно передать данные в процедуру, и любые изменения параметра будут возвращены вызывающей процедуре.

- С помощью выходного параметра нельзя передать данные в процедуру; он используется только для передачи данных обратно вызывающей стороне.
- С помощью выходного параметра нельзя передать данные в процедуру, а также невозможно получить обратно данные из процедуры из выходного параметра.

Ответы

Закрепление материала

1. Правильные ответы: А и С.
 - A. **Правильно:** инструкция WHILE запускает циклическую структуру.
 - B. **Неправильно:** BEGIN и END не вызывают ветвления, они используются только для группирования инструкций.
 - C. **Правильно:** IF и ELSE вызывают ветвление выполнения кода на основе условия, указанного в предложении IF.
 - D. **Неправильно:** инструкция GO только прерывает пакет. Сама по себе она не влияет на выполнение кода.
2. Правильный ответ: С.
 - A. **Неправильно:** переменные вызывающей процедуры не могут быть видны вызываемой процедуре.
 - B. **Неправильно:** временные таблицы видны, но передаваемые параметры также видны.
 - C. **Правильно:** вызываемая процедура видит временные таблицы и параметры, передаваемые ей вызывающей процедурой.
 - D. **Неправильно:** вызываемая процедура может видеть временные таблицы и передаваемые параметры из вызывающей процедуры.
3. Правильный ответ: В.
 - A. **Неправильно:** можно использовать выходной параметр для возвращения информации из хранимой процедуры.
 - B. **Правильно:** вы можете передавать данные в хранимую процедуру и извлекать информацию из нее с помощью выходного параметра.
 - C. **Неправильно:** выходной параметр используется не только для передачи данных обратно вызывающей стороне хранимой процедуры. Он также используется для передачи данных от вызывающей стороны в хранимую процедуру.
 - D. **Неправильно:** вы можете передавать данные в хранимую процедуру и извлекать информацию из нее с помощью выходного параметра.

Занятие 2

13.2 Реализация триггеров

Триггер — это особый тип хранимой процедуры, связанный с выбранными событиями языка обработки данных в таблице или представлении. Триггер не может быть выполнен явно. Он срабатывает, когда запускает событие DML, связанное с этим триггером, такое как инструкции INSERT, UPDATE или DELETE.

13.2.1 Триггеры DML

Триггер DML — это пакет T-SQL, связанный с таблицей, которая предназначена для того, чтобы отвечать на определенное событие DML, такое как инструкция INSERT, UPDATE или DELETE, или комбинацию этих событий.

- AFTER — этот триггер срабатывает после того, как событие, с которым он связан, завершается, и может быть определен только для постоянных таблиц;
- INSTEAD OF — этот триггер срабатывает вместо события, с которым он связан, и может быть определен в постоянных таблицах и представлениях.

Оба типа триггеров DML выполняются как часть транзакции, связанной с инструкцией INSERT, UPDATE или DELETE. Триггер рассматривается как часть транзакции, которая включает событие, вызывающее срабатывание триггера.

13.2.2 Триггеры AFTER

```
1 CREATE TRIGGER Sales.tr_SalesOrderDetailsDML
2 ON Sales.OrderDetails
3 FOR DELETE, INSERT, UPDATE
4 AS
5 BEGIN
6     SET NOCOUNT ON
```

Прежде всего, убедитесь в том, что это триггер AFTER. В определении триггера типом по умолчанию является AFTER, если указан элемент FOR. Но элемент FOR может быть заменен либо AFTER, либо INSTEAD OF для определения типа триггера.

13.2.3 Оптимизация триггера

СОВЕТ

Подготовка к экзамену

Когда выполняются инструкции INSERT, UPDATE или DELETE и не обрабатываются никакие строки, нет смысла в продолжении работы триггера. Можно повысить производительность триггера, выполняя проверку @@ROWCOUNT на равенство 0 в самой первой строке триггера. Это должна быть первая строка, поскольку @@ROWCOUNT будет переустановлена в 0 любой дополнительной инструкцией. Когда триггер AFTER начнет работать, @@ROWCOUNT будет содержать число строк, задействованных внешней инструкцией INSERT, UPDATE или DELETE.

```

1 CREATE TRIGGER Sales.tr_SalesOrderDetailsDML
2   ON Sales.OrderDetails
3   AFTER DELETE ,  INSERT ,  UPDATE
4   AS
5   BEGIN
6     IF @@ROWCOUNT = 0 RETURN ;
7     SET NOCOUNT ON ;
8     SELECT COUNT(*) AS InsertedCount FROM Inserted ;
9     SELECT COUNT(*) AS DeletedCount FROM Deleted ;
10    END ;

```

СОВЕТ

Подготовка к экзамену

Считается не лучшим решением возвращать результирующие наборы из триггеров. В SQL Server 2012 и более ранних версиях возвращение набора строк из триггера разрешено, но на него не следует полагаться. Эту опцию можно запретить с помощью параметра хранимой процедуры sp_configure, который называется **Disallow Results From Triggers** (Запретить результаты триггеров). Кроме того, возможность возвращать результирующие наборы из триггера является устаревшей и будет удалена в версии SQL Server, следующей за SQL Server 2012.

13.2.4 Триггеры INSTEAD OF

Хотя триггеры INSTEAD OF могут создаваться как для таблиц, так и для представлений, обычно они используются с представлениями. Причина в том, что когда инструкция UPDATE отправляется к представлению, может быть обновлена только одна таблица за один раз. Кроме того, в представлении могут быть агрегаты функций на столбцах, не допускающие прямого обновления. Триггер INSTEAD OF может взять инструкцию UPDATE применительно к представлению и, вместо ее выполнения, заменить ее двумя инструкциями UPDATE применительно к базовой таблице представления.

```
1 CREATE TRIGGER Production.tr_ProductionCategories_categoryname
2 ON Production.Categories
3 INSTEAD OF INSERT
4 AS
5 BEGIN
6     SET NOCOUNT ON;
7     IF EXISTS (SELECT COUNT(*)
8         FROM Inserted AS I
9         LEFT JOIN Production.Categories AS C
10            ON I.categoryname = C.categoryname
11        GROUP BY I.categoryname
12        HAVING COUNT(*) > 0 )
13     BEGIN
14         THROW 50000, 'Duplicate category names not allowed', 0;
15     END;
16     ELSE
17         INSERT Production.Categories (categoryname, description)
18             SELECT categoryname, description FROM Inserted;
19     END;
```

13.2.5 Функции триггеров DML

Для получения информации о том, что происходит в коде, можно использовать в триггере две функции.

- UPDATE(). Эту функцию можно использовать, чтобы определить, имеет ли конкретный столбец ссылки от инструкций INSERT или UPDATE.

Например, можно вставить в триггер следующий код: IF UPDATE(qty)
PRINT 'Column qty affected';

- COLUMNS_UPDATED(). Можно использовать эту функцию, если известен последовательный номер столбца в таблице. Вы должны будете использовать битовую операцию И чтобы проверить, был ли столбец обновлен.

КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Какие два типа триггеров DML могут быть созданы?
2. Если триггер AFTER обнаруживает ошибку, как он препятствует завершению команды?

Ответы на контрольные вопросы

1. Можно создать два типа триггеров DML — AFTER и INSTEAD OF.
2. Триггер AFTER запускает команду THROW или RAISERROR, чтобы вызвать откат транзакции DML-команды.

Резюме занятия

- Триггер DML — это пакет кода T-SQL, подобный хранимой процедуре, который связан с таблицей и иногда с представлением. Триггеры DML могут использоваться для аудита, реализации сложных правил целостности и т. п.
- Триггеры выполняются, когда происходит определенное событие DML, такое как инструкция INSERT, UPDATE или DELETE.
- SQL Server поддерживает два типа триггеров DML: триггеры AFTER и триггеры INSTEAD OF. Триггеры DML обоих типов выполняются как часть транзакции, связанной с инструкциями INSERT, UPDATE или DELETE.

- В коде T-SQL для обоих типов триггеров DML можно получить доступ к таблицам, называемым вставленными и удаленными таблицами. Эти таблицы содержат строки, модификация которых приводила к запуску триггера.

Закрепление материала

Закрепление материала

Для проверки знания материала, изложенного в данном занятии, ответьте на следующие вопросы. Ответы на эти вопросы и пояснения, почему тот или иной ответ верен или неверен, можно найти в *приложении "Ответы" в конце книги*.

1. Как работают вставленные и удаленные таблицы с инструкцией DML в триггере AFTER?
 - A. При использовании инструкции `DELETE` вставленная таблица содержит новые строки, а удаленная таблица — удаленные строки.
 - B. Вставленная таблица содержит только строки из инструкции `INSERT`, а удаленная таблица — только строки из инструкции `DELETE`.
 - C. При использовании инструкции `INSERT` вставленная таблица содержит новые строки, а удаленная таблица остается пустой.
 - D. При использовании инструкции `UPDATE` вставленная таблица остается пустой, а удаленная таблица содержит все измененные строки.
2. Какие из перечисленных утверждений справедливы для триггера `INSTEAD OF`? (Назовите все возможные варианты.)
 - A. Триггеры `INSTEAD OF` можно открывать на представлениях.
 - B. Триггеры `INSTEAD OF` выполняются вместо триггеров `AFTER`.
 - C. Триггеры `INSTEAD OF` могут быть объявлены только для инструкций `UPDATE`.
 - D. Триггеры `INSTEAD OF` выполняют код вместо исходной инструкции DML.
3. Как можно отключить встроенные триггеры на экземпляре SQL Server с помощью T-SQL?
 - A. Нужно использовать хранимую процедуру `sp_configure` перед '`nested triggers`' и '`OFF`'.
 - B. Нужно использовать хранимую процедуру `sp_configure` перед '`nested triggers`' и `0`.
 - C. Нужно использовать хранимую процедуру `sp_configure` перед '`nested triggers`' и '`OFF`', после которых следует инструкция `RECONFIGURE`.
 - D. Нужно использовать хранимую процедуру `sp_configure` перед '`nested triggers`' и `0`, после которых следует инструкция `RECONFIGURE`.

Ответы

-
1. Правильный ответ: С.
 - A. **Неправильно:** в случае инструкции `DELETE` вставленная таблица будет пустой, поскольку нет новых или измененных строк.

- B. **Неправильно:** вставленная и удаленная таблицы содержат строки не только для инструкций `INSERT` и `DELETE`, но и для инструкции `UPDATE`.
 - C. **Правильно:** инструкция `INSERT` содержит все вставленные строки во вставленной таблице и не содержит строк в удаленной таблице.
 - D. **Неправильно:** в случае инструкции `UPDATE`, которая обновляет строки в таблице, изменяемые строки появятся во вставленной таблице со своими новыми значениями, и в удаленной таблице со своими старыми значениями.
2. Правильные ответы: А и D.
 - A. **Правильно:** вы можете создать триггеры `INSTEAD OF` на представлениях для перенаправления вставок или обновлений в базовые таблицы.
 - B. **Неправильно:** триггеры `INSTEAD OF` выполняются вместо инструкций DML, а не вместо триггеров `AFTER`.
 - C. **Неправильно:** триггеры `INSTEAD OF` могут быть объявлены для всех инструкций DML — `INSERT`, `UPDATE` и `DELETE`.
 - D. **Правильно:** используя триггеры `INSTEAD OF`, можно подставить код триггера вместо исходной инструкции DML.
 3. Правильный ответ: D.
 - A. **Неправильно:** `OFF` не является допустимым значением второго параметра хранимой процедуры `sp_configure`.
 - B. **Неправильно:** Для параметра '`nested triggers`' требуется дополнительная инструкция `RECONFIGURE`.
 - C. **Неправильно:** не является допустимым значением второго параметра хранимой процедуры `sp_configure`.
 - D. **Правильно:** После хранимой процедуры `sp_configure`, стоящей перед '`nested triggers`' и 0, вы должны также выполнить инструкцию `RECONFIGURE`.

13.3 Основные сведения об определяемых пользователем функциях

В отличие от хранимых процедур, определяемые пользователем функции встроены в инструкции T-SQL и выполняются как часть команды T-SQL. Определяемые пользователем функции не могут выполняться с помощью команды EXECUTE.

Определяемые пользователем функции имеют доступ к данным SQL Server, но не могут выполнять DDL, т. е. они не могут создавать таблицы, а также модифицировать таблицы, индексы или любые другие объекты или изменять любые данные в постоянных таблицах с помощью инструкций DML.

13.3.1 Скалярные определяемые пользователем функции

```
1 CREATE FUNCTION dbo.FunctionName
2   ( @param1 int ,
3     @param2 int )
4 RETURNS INT
5 AS
6 BEGIN
7   RETURN @param1 + @param2
8 END
```

13.3.2 Встроенная пользовательская функция с табличным значением

```
1 CREATE FUNCTION dbo.FunctionName
2   ( @param1 int ,
3     @param2 char(5) )
4 RETURNS TABLE AS RETURN
5   ( SELECT @param1 AS c1 ,
6     @param2 AS c2 )
```

13.3.3 Многооператорная пользовательская функция с табличным значением

```
1 CREATE FUNCTION dbo.FunctionName
2   ( @param1 int,
3     @param2 char(5) )
4 RETURNS @returntable TABLE
5   ( c1 int,
6     c2 char(5) )
7 AS
8 BEGIN
9   INSERT @returntable
10  SELECT @param1, @param2
11 RETURN
12 END;
```

13.3.4 Ограничения для определяемых пользователем функций

Определяемые пользователем функции не могут выполнять следующие действия:

- применять какие-либо изменения к схеме или данным в базе данных;
- изменять состояние базы данных или экземпляра SQL Server;
- создавать временные таблицы или иметь к ним доступ;
- вызывать хранимые процедуры;
- выполнять динамический SQL;

- производить побочные эффекты. Например, функции RAND() и NEWID() используют информацию из предыдущего вызова. Использование предыдущей информации является "побочным эффектом что недопустимо.
-

КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Назовите два типа определяемых пользователем функций с табличным значением.
2. Какой тип определяемой пользователем функции возвращает всего одно значение?

Ответы на контрольные вопросы

1. Можно создавать встроенные или многооператорные определяемые пользователем функции с табличным значением.
2. Скалярная определяемая пользователем функция возвращает только одно значение.

Резюме занятия

- Определяемые пользователем функции инкапсулируют повторно используемый код T-SQL и возвращают вызывающей стороне скалярное значение или таблицу.
- Как и хранимые процедуры, определяемые пользователем функции могут принимать параметры, доступ к которым можно получить внутри функции как к переменным. В отличие от хранимых процедур, определяемые пользователем функции встроены в инструкции T-SQL и выполняются как часть команды T-SQL. Определяемые пользователем функции нельзя выполнять с помощью команды EXECUTE.
- Определяемые пользователем функции имеют доступ к данным SQL Server, но не могут выполнять никаких DDL, т. е. они не могут выполнять модификацию таблиц, индексов или других объектов, а также изменять таблицы данных с помощью DML.
- Существуют два основных типа определяемых пользователем функций: скалярные и возвращающие табличное значение. Пользовательская ска-

ларная функция возвращает вызывающей стороне одно значение и может вызываться из множества мест, включая список SELECT и предложение WHERE. Функция, возвращающая табличное значение, возвращает таблицу и может появляться в предложении FROM. И определяемые пользователем скалярные функции, и определяемые пользователем функции с табличным значением могут состоять из одной или нескольких строк кода T-SQL.

Закрепление материала

Закрепление материала

Для проверки знания материала, изложенного в данном занятии, ответьте на следующие вопросы. Ответы на эти вопросы и пояснения, почему тот или иной ответ верен или неверен, можно найти в *приложении "Ответы" в конце книги*.

1. Что из перечисленного далее справедливо для скалярной пользовательской функции?
 - A. Пользовательские скалярные функции являются и встроенными, и многооператорными.
 - B. Пользовательские скалярные функции возвращают результат инструкции SELECT.
 - C. Пользовательские скалярные функции могут вызываться в списке SELECT или предложении WHERE.
 - D. Пользовательские скалярные функции могут вызываться в предложении FROM инструкции SELECT.
2. Что из перечисленного далее справедливо для определяемых пользователем функций с табличным значением?
 - A. Определяемые пользователем функции с табличным значением могут возвращать скалярные значения или таблицы.
 - B. Определяемые пользователем функции с табличным значением всегда работают с несколькими инструкциями T-SQL.
 - C. Определяемые пользователем функции с табличным значением могут вызываться в списке SELECT или предложении WHERE.
 - D. Определяемые пользователем функции с табличным значением могут вызываться в предложении FROM инструкции SELECT.
3. Какое из высказываний лучше всего описывает разницу между встроенной пользовательской функцией с табличным значением и многооператорной пользовательской функцией с табличным значением?
 - A. Встроенная пользовательская функция с табличным значением определяет схему табличной переменной с именами столбцов и типами данных и вставляет данные в табличную переменную.
 - B. Встроенная пользовательская функция с табличным значением определяет схему постоянной таблицы с именами столбцов и типами данных и вставляет данные в эту таблицу.
 - C. Многооператорная пользовательская функция с табличным значением определяет схему табличной переменной с именами столбцов и типами данных и вставляет данные в табличную переменную.
 - D. Многооператорная пользовательская функция с табличным значением определяет схему постоянной таблицы с именами столбцов и типами данных и вставляет данные в эту таблицу.

Ответы

Закрепление материала

1. Правильный ответ: С.
 - A. **Неправильно:** пользовательские скалярные функции никогда не бывают встроенными. Только определяемые пользователем функции с табличным значением могут быть встроенными.
 - B. **Неправильно:** результатом инструкции SELECT может быть таблица, а пользовательские скалярные функции не могут возвращать таблицу.
 - C. **Правильно:** можно вызвать пользовательскую скалярную функцию в списке SELECT или в условном предложении WHERE, в любом месте, где скалярное значение применимо.
 - D. **Неправильно:** предложение FROM требует таблицу, а пользовательские скалярные функции не могут возвращать таблицу.

682

Ответы

2. Правильный ответ: D.
 - A. **Неправильно:** определяемые пользователем функции с табличным значением возвращают только таблицы.
 - B. **Неправильно:** встроенные определяемые пользователем функции с табличным значением состоят только из одной инструкции T-SQL. Даже многооператорные определяемые пользователем функции с табличным значением требуют только одной инструкции T-SQL.
 - C. **Неправильно:** вызов в списке SELECT или предложении WHERE будет требовать скалярного значения, а определяемые пользователем функции с табличным значением возвращают только таблицы.
 - D. **Правильно:** предложение FROM требует таблицу, и определяемые пользователем функции с табличным значением возвращают таблицы.
3. Правильный ответ: C.
 - A. **Неправильно:** встроенная пользовательская функция с табличным значением не определяет схему табличной структуры, которую она возвращает.
 - B. **Неправильно:** встроенная пользовательская функция с табличным значением не может создать постоянную таблицу.
 - C. **Правильно:** многооператорная пользовательская функция с табличным значением определяет явную схему табличной переменной и затем вставляет данные в эту табличную переменную.
 - D. **Неправильно:** многооператорная пользовательская функция с табличным значением не может создать постоянную таблицу.

Упражнения

Упражнения

Ответы

Упражнение 1. Реализация хранимых процедур и определяемых пользователем функций

Вы назначены на новый проект. Как ведущий разработчик баз данных, вы заметили, что почти вся проверка данных в базе данных происходит в клиентских программах. Иногда неустранимые ошибки в клиентском программном обеспечении вызывают несогласованность базы данных, и вы хотите провести рефакторинг системы с помощью хранимых процедур, чтобы помочь защитить базу данных. Ответьте на следующие вопросы о том, какие действия вы можете предпринять для увеличения надежности приложения.

1. Какие шаги можно предпринять для предотвращения появления дубликатов или несогласованностей уникальных ключей и несоответствующих внешних ключей?
2. Как вы можете представить стандартный интерфейс кода приложения к базе данных?
3. Разработчики клиента хотят поместить параметры в представления, но T-SQL не позволяет им это сделать. Что вы можете использовать вместо параметризованных представлений?
4. Существует одна большая таблица, в которой часто выполняется поиск по трем разным столбцам, но пользователь может выбрать любой из столбцов и оставить остальные пустыми. Как можно использовать хранимые процедуры для увеличения эффективности этого поиска?

Упражнение 2. Реализация триггеров

Вас попросили проверить код T-SQL существующего приложения базы данных и дать рекомендации по его усовершенствованию. Ответьте на следующие вопросы по поводу рекомендаций, которые вы можете дать по данному проекту.

1. Вы заметили, что система использует множество триггеров для обеспечения ограничений внешнего ключа, триггеры подвержены ошибкам и их трудно отлаживать. Какие изменения для уменьшения использования триггеров вы можете порекомендовать?
2. Вы также заметили, что имеются сложные операции, использующие вложенные триггеры, которые невозможно заставить работать в приложении корректно. Какие действия вы можете порекомендовать, чтобы избавиться от использования вложенных триггеров?

3. Приложение должно часто выполнять вставку данных в главную таблицу и в несколько вспомогательных таблиц в одном и том же действии, делая код приложения очень сложным. Что вы можете порекомендовать в качестве способа переместить части сложности из приложения в базу данных?
4. Имеется важная таблица, которая при каждом изменении данных требует простых действий по регистрации этих изменений в журнале. Ведение журнала выполняется в настраиваемой таблице, построенной специально, чтобы удовлетворять требованиям приложения. Какие вы можете дать рекомендации для реализации такого ведения журнала?

1. Чтобы предотвратить несогласованность базы данных, следите за тем, чтобы были установлены нужные ограничения: ограничения первичного и уникального ключей на таблицах, проверочные ограничения на столбцах и ограничения внешнего ключа между таблицами. Прочие более сложные бизнес-правила можно обеспечить с помощью триггеров.
2. Чтобы представить стандартный интерфейс к базе данных, используйте хранимую процедуру уровня данных, т. е. используйте стандартные хранимые процедуры вставки, обновления и удаления для каждой таблицы. Клиентское программное обеспечение должно только изменять данные в таблицах с помощью этих хранимых процедур.
3. Вы можете использовать функции, возвращающие табличное значение, вместо представлений и определить параметры для удовлетворения требований разработчиков приложения. Затем вы можете вызвать функцию изнутри хранимой процедуры, которая принимает эти параметры и отправляет результаты обратно клиенту.

4. Рассмотрите возможность создания хранимой процедуры поиска, которая состоит из драйвера, и заставьте ее вызывать вложенные процедуры, по одной для каждой комбинации параметра. Эти вложенные процедуры всегда будут иметь тот же план запроса, поэтому перекомпиляция процедур не потребуется.

Упражнение 2. Реализация триггеров

1. Ограничения внешнего ключа можно реализовать с помощью триггеров, но код может стать сложным и ненадежным. Вместо того вы можете порекомендовать разработчикам базы данных реализовать ссылочную целостность с помощью объявленных ограничений внешнего ключа T-SQL вместо триггеров.
2. Вы можете рекомендовать, чтобы приложение запретило вложенные триггеры на сервере разработки, так чтобы разработчики баз данных привыкли к выполнению всех необходимых действий в пределах одного уровня триггера. Это поможет упростить код триггера и повысить возможность его отладки.
3. Когда приложение вставляет данные в одну таблицу и должно также выполнять вставку во вспомогательные таблицы в рамках того же действия, вы можете порекомендовать разработчикам базы данных использовать для этого триггер `INSTEAD OF`. В этом триггере несколько вставок могут выполняться перед вставкой в главную таблицу.
4. Для поддержки простого ведения журнала вы можете рекомендовать разработчикам использовать DML-триггер `AFTER`. Триггер этого типа выполняется после инструкций `INSERT`, `UPDATE` или `DELETE` и может осуществлять запись в таблицу ведения журнала.

14 Использование инструментов анализа производительности запросов

14.1 Основные понятия оптимизации запросов

14.1.1 Проблемы оптимизации запросов и оптимизатор запросов

SQL Server может выполнять объединение разными способами. Он может использовать любой из следующих алгоритмов объединения:

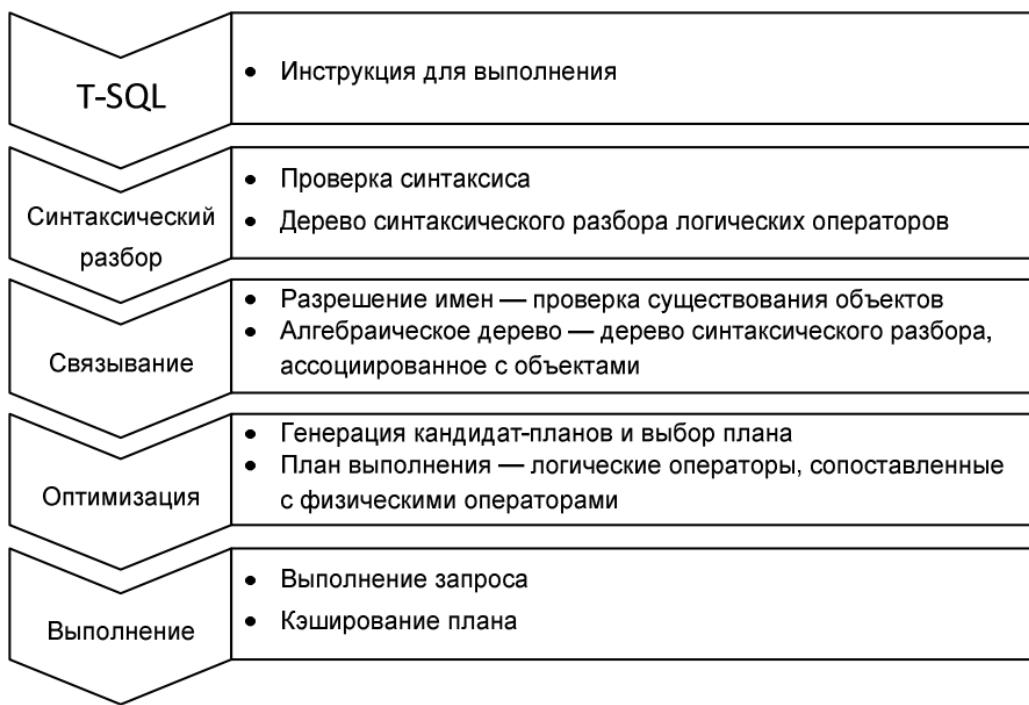
- вложенные циклы (Nested Loops);
- слияние (Merge);
- хэш (Hash);
- хэш оптимизированной фильтрации по битовым картам (Bitmap Filtering Optimized Hash, также называемый оптимизацией типа "звезда").

Прежде чем продолжить рассматривать процесс оптимизации запросов, в данном занятии мы кратко опишем, как SQL Server выполняет запросы.

Сниффинг параметров — это процесс, когда SQL Server пытается перехватить (sniff) значение текущего параметра в процессе компиляции и передает его оптимизатору запросов.

SQL Server проделывает огромную работу по оптимизации запросов и поддерживает кэшированные планы. Но существует множество ситуаций, когда что-то идет не так, как нужно, и выбирается не лучший план, как, например, в следующих сценариях:

- выбран не лучший план, поскольку область поиска планов выполнения была слишком большой;



- статистическая информация не представлена или не обновлена, что ведет к неправильной оценке мощности;
- кэшированный план является неоптимальным для данного значения параметра;
- перехват параметров ведет к неточной оценке мощности;
- оптимизатор запросов недооценивает или переоценивает стоимость алгоритма, реализованного в физическом операторе;
- изменения в аппаратном обеспечении могут больше соответствовать другому плану выполнения. Например, кто-то мог добавить в блок процессоры, и план, который использует большее время центрального процессора (ЦП), может быть более подходящим.

КОНТРОЛЬНЫЙ ВОПРОС

- Что является результатом этапа синтаксического разбора выполнения запроса?

Ответ на контрольный вопрос

- Результатом этого этапа, при условии, что запрос прошел проверку синтаксиса, является дерево логических операций, известное как дерево синтаксического разбора.

14.1.2 Подсистема расширенных событий SQL Server, трассировка SQL и приложение SQL Server Profiler

Пакет расширенных событий представляет собой контейнер всех объектов подсистемы расширенных событий. К этим объектам относятся следующие.

- **События (Events).** Это точки мониторинга, которые вас интересуют. Можно использовать события для мониторинга или для запуска синхронных или асинхронных действий.
- **Цели (Targets).** Это получатели событий. Можно использовать цели, которые выполняют запись в файл, хранят данные события в буфере памяти или собирают статистические данные о событиях. Цели могут выполнять синхронную или асинхронную обработку данных.
- **Действия (Actions).** Это ответы на события. Они привязаны к событию. Действия могут получать копию стека и проверять данные, сохранять информацию в локальной переменной, собирать статистические данные о событиях и даже добавлять данные к данным события. Например, в SQL Server можно использовать действие обнаружения плана выполнения для нахождения плана выполнения.
- **Предикаты (Predicates).** Это наборы логических правил для фильтрации собранных событий. Для минимизации влияния сессии мониторинга на систему важно захватывать только нужные события.
- **Типы (Types).** Они помогают интерпретировать собранные данные. В действительности, данные — это коллекция байтов, а типы представляют контекст данных. Тип имеют события, действия, цели, предикаты и сами типы.

- **Карты (Maps).** Это внутренние таблицы SQL Server, которые отображают внутренние числовые значения на значимые строки.

Трассировка SQL (SQL Trace) — это встроенный механизм для получения событий.

СОВЕТ

Подготовка к экзамену

Следует использовать подсистему расширенных событий SQL вместо трассировки и приложения SQL Server Profiler, поскольку подсистема расширенных событий является более легкой, а также потому что трассировка SQL и приложение SQL Server Profiler не будут использоваться в следующих версиях SQL Server.

Резюме занятия

- Оптимизатор запросов генерирует кандидат-планы выполнения и оценивает их.
- SQL Server предоставляет множество инструментов, которые помогают пользователям анализировать запросы, включая подсистему расширенных событий, трассировку SQL и приложение SQL Server Profiler.
- Подсистема расширенных событий является более легким механизмом мониторинга, чем трассировка SQL.
- SQL Server Profiler предоставляет пользовательский графический интерфейс для доступа к трассировке SQL.

Закрепление материала

1. Какие действия относятся к этапу оптимизации выполнения запроса? (Выберите все подходящие варианты.)
 - A. Генерация алгебраического дерева.
 - B. Генерация кандидат-планов.
 - C. Выбор лучшего кандидат-плана.
 - D. Кэширование плана.
 - E. Выполнение запроса.
2. На каком этапе выполнения запроса SQL Server проверяет, существуют ли объекты, на которые ссылается запрос?
 - A. На этапе грамматического разбора.
 - B. На этапе привязки.
 - C. На этапе оптимизации.
 - D. На этапе выполнения.
3. Что из перечисленного не является частью пакета расширенных событий?
 - A. Предикаты.
 - B. Цели.
 - C. Источники.
 - D. Действия.

Ответы

Закрепление материала

1. Правильные ответы: В и С.
 - A. **Неправильно:** алгебраическое дерево генерируется на этапе связывания.
 - B. **Правильно:** на этапе оптимизации SQL Server генерирует кандидат-планы.
 - C. **Правильно:** на этапе оптимизации SQL Server выбирает план выполнения из коллекции кандидат-планов.
 - D. **Неправильно:** план кэшируется на этапе выполнения.
 - E. **Неправильно:** запрос выполняется на этапе выполнения.
2. Правильный ответ: В.
 - A. **Неправильно:** на этапе синтаксического разбора SQL Server выполняет проверку на синтаксическую корректность.

684

Ответы

- В. **Правильно:** SQL Server выполняет разрешение имен объектов и их связывание в логические операторы на этапе связывания.
 - С. **Неправильно:** на этапе оптимизации SQL Server генерирует кандидат-планы и выбирает план выполнения.
 - Д. **Неправильно:** на этапе выполнения SQL Server выполняет запрос и кэширует план выполнения.
3. Правильный ответ: С.
 - A. **Неправильно:** предикаты — это объекты пакета расширенных событий.
 - B. **Неправильно:** цели — это объекты пакета расширенных событий.
 - C. **Правильно:** источники не входят в состав пакета расширенных событий.
 - D. **Неправильно:** действия — это объекты пакета расширенных событий.

14.2 Использование параметров сеанса инструкции SET и анализ планов запросов

14.2.1 Параметры сеанса инструкции SET

SQL Server хранит данные на страницах. Страница — это физическая единица на диске внутри базы данных SQL Server. Страница имеет фиксированный размер, равный 8192 байт, или 8 Кбайт. Страница принадлежит только одному объекту, т. е. одной таблице, индексу или индексированному представлению. Далее страницы группируются в логические группы по 8 страниц, называемые экстентами. Экстент может быть смешанным, если страницы в этом экстенте принадлежат нескольким объектам, или однородным, когда все страницы этого экстента принадлежат только одному объекту.

Следующий код проверяет число страниц, которые таблицы Sales.Customers и Sales.Orders занимают в базе данных TSQL2012.

```
1 DBCC DROPCLEANBUFFERS;
2 SET STATISTICS IO ON;
3 SELECT * FROM Sales.Customers;
4 SELECT * FROM Sales.Orders;
```

Обратите внимание, инструкция DBCC DROPCLEANBUFFERS удаляет данные из кэша.

Далее приведены обозначения, используемые в возвращенной запросом информации:

- scan count (число просмотров) — количество выполненных просмотров таблицы или индекса;
- logical reads (логических чтений) — количество страниц, считанных из кэша данных. Когда выполняется чтение целой таблицы, как в запросах из приведенного примера, это число дает оценку размера таблицы;
- physical reads (физических чтений) — количество страниц, считанных с диска. Это число меньше, чем фактическое количество страниц, поскольку множество страниц находится в кэше;

- read-ahead reads (упреждающих чтений) — количество страниц, прочитанных SQL Server с упреждением;
- lob logical reads (LOB логических чтений) — число страниц больших объектов (large object page, LOB), считанных из кэша данных. К большим объектам относятся столбцы типов VARCHAR(MAX), NVARCHAR(MAX), VARBINARY(MAX), TEXT, NTEXT, IMAGE, XML или большие типы данных CLR, включая системные пространственные CLR-типы GEOMETRY и GEOGRAPHY;
- lob physical reads (LOB физических чтений) — число страниц типов больших объектов, считанных с диска;
- lob read-ahead reads (LOB упреждающих чтений) — число страниц типов больших объектов, прочитанных SQL Server с упреждением.

Следующая полезная для анализа производительности команда SET уровня сеанса — это команда SET STATISTICS TIME.

```

1  SET STATISTICS TIME ON;
2  DBCC DROPCLEANBUFFERS;
3
4  SELECT C.custid, C.companyname,
5      O.orderid, O.orderdate
6  FROM Sales.Customers AS C
7  INNER JOIN Sales.Orders AS O
8  ON C.custid = O.custid;

```

14.2.2 Планы выполнения

На плане выполнения можно увидеть физические операторы, используемые в процессе выполнения.

Резюме занятия

- Можно использовать параметры инструкции SET уровня сеанса для анализа запросов.

КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Как вы можете быстро измерить количество дисковых операций ввода-вывода, которые выполняет запрос?
2. Как вы можете получить предполагаемый план выполнения в формате XML для дальнейшего анализа?

Ответы на контрольные вопросы

1. Вам следует использовать команду `SET STATISTICS IO`.
2. Вы можете использовать команду `SET SHOWPLAN_XML`.

- Можно использовать графические планы выполнения для получения подробных сведений о том, как SQL Server выполняет запрос.
- Можно получить отображение предварительного и действительного плана выполнения.
- В графическом плане выполнения можно найти подробные свойства каждого оператора.

Закрепление материала

Более полные пояснения, можно найти в приложении «Ответы» в конце книги.

1. Какие параметры инструкции SET уровня сеанса полезны для оптимизации запросов? (Выберите все подходящие ответы.)

- A. SET STATISTICS IO.
 - B. SET STATISTICS EXECUTION_DETAILS.
 - C. SET IDENTITY_INSERT.
 - D. SET STATISTICS.
2. Как можно прочесть графический план выполнения?
 - A. Сверху вниз, слева направо.
 - B. Сверху вниз, справа налево.
 - C. Слева направо, сверху вниз.
 - D. Справа налево, сверху вниз.
 3. Какие команды включают XML-план?
 - A. SET EXECUTION_XML ON.
 - B. SET SHOWPLAN_XML ON.
 - C. SET XML PLAN ON.
 - D. SET STATISTICS XML ON.

Ответы

1. Правильные ответы: А и D.
 - A. **Правильно:** параметр сеанса SET STATISTICS IO полезен для анализа производительности запроса.
 - B. **Неправильно:** не существует параметра SET STATISTICS EXECUTION_DETAILS.
 - C. **Неправильно:** параметр SET IDENTITY_INSERT используется для предоставления значения для столбца, имеющего свойство идентификатора.
 - D. **Правильно:** параметр сеанса SET STATISTICS TIME полезен для анализа производительности запроса.
2. Правильный ответ: D.
 - A. **Неправильно:** планы выполнения следует читать справа налево, сверху вниз.
 - B. **Неправильно:** планы выполнения следует читать справа налево, сверху вниз.
 - C. **Неправильно:** планы выполнения следует читать справа налево, сверху вниз.
 - D. **Правильно:** планы выполнения следует читать справа налево, сверху вниз.
3. Правильные ответы: В и D.
 - A. **Неправильно:** не существует команды SET EXECUTION_XML.
 - B. **Правильно:** команда SET SHOWPLAN_XML используется для включения предполагаемых XML-планов.
 - C. **Неправильно:** не существует команды SET XML PLAN.
 - D. **Правильно:** команда SET STATISTICS XML используется для включения фактических XML-планов.

14.3 Использование динамических административных объектов

14.3.1 Введение в динамические административные объекты

Динамические административные объекты не сохраняются ни в какой базе данных; динамические административные объекты — это виртуальные объекты, которые предоставляют вам доступ к собранным SQL Server данным в памяти.

14.3.2 Наиболее важные динамические административные объекты для настройки объектов

Вы можете начать сеанс анализа со сбора системной информации. Используя относящееся к SQL Server динамическое административное представление sys dm_os_sys_info, можно получить основную информацию об экземпляре сервера, как показано в следующем запросе:

```
1  SELECT cpu_count AS logical_cpu_count,
2      cpu_count / hyperthread_ratio AS physical_cpu_count,
3      CAST(physical_memory_kb / 1024. AS int) AS physical_memory_mb,
4      sqlserver_start_time
5  FROM sys.dm_os_sys_info;
```

Этот запрос возвращает информацию о количестве логических ЦП, физических ЦП, физической памяти и времени запуска SQL Server. Последняя информация говорит о том, имеет ли смысл анализировать накопленную информацию или нет.

Относящееся к SQL Server динамическое административное представление sys dm_os_waiting_tasks предоставляет информацию о сессиях, которые в данный момент ожидают чего-либо.

```
1  SELECT S.login_name, S.host_name, S.program_name,
2      WT.session_id, WT.wait_duration_ms,
3      WT.wait_type, WT.blocking_session_id, WT.resource_description
4  FROM sys.dm_os_waiting_tasks AS WT
5  INNER JOIN sys.dm_exec_sessions AS S
6  ON WT.session_id = S.session_id
7  WHERE s.is_user_process = 1;
```

Связанное с выполнением запросов динамическое административное представление sys dm_exec_requests возвращает информацию о выполняющихся в данный момент запросах.

```
1  SELECT S.login_name, S.host_name, S.program_name,
2      R.command, T.text,
3      R.wait_type, R.wait_time, R.blocking_session_id
4  FROM sys.dm_exec_requests AS R
5  INNER JOIN sys.dm_exec_sessions AS S
6  ON R.session_id = S.session_id
```

```
7 |     OUTER APPLY sys.dm_exec_sql_text(R.sql_handle) AS T  
8 | WHERE S.is_user_process = 1;
```

Вы можете извлечь множество информации о выполняемых запросах из связанного с выполнением запросов динамического административного представления sys dm_exec_query_stats.

Резюме занятия

- Динамические административные объекты помогают быстро получить информацию, собранную SQL Server.
- Для анализа запросов следует использовать SQLOS, относящиеся к выполнению, и связанные с индексом динамические административные объекты.
- Связанные с индексом динамические административные объекты не только предоставляют полезные сведения об использовании индексов, они также информируют об отсутствующих индексах.

Закрепление материала

1. Какой динамический административный объект предоставляет информацию об использовании индексов?
 - A. `SYS.dm_exec_query_stats`.
 - B. `sys.dm_exec_query_text`.
 - C. `sys.dm_db_index_usage_stats`.
 - D. `sys.indexes`.
2. Какой недостаток динамических управляющих объектов считается наиболее важным?
 - A. Необходимость иметь достаточное количество собранных данных с момента последнего перезапуска SQL Server.
 - B. Динамические управляющие объекты сложно использовать.
 - C. Динамические управляющие объекты недоступны в стандартной редакции SQL Server.
 - D. Необходимость повторно создавать динамические управляющие объекты перед каждым сбором аналитики.
3. Как вы можете найти текст выполненного запроса с помощью динамических управляющих объектов?
 - A. Эта информация представлена в динамическом административном представлении `sys.dm_exec_query_stats`.
 - B. Выполнив запрос к динамическому административному представлению `sys.dm_exec_query_stats`.

- C. Динамическое административное представление `sys.dm_exec_query_stats` возвращает текст запроса.
- D. Невозможно найти текст запроса с помощью динамических управляющих объектов.

Ответы

Занятие 3

Закрепление материала

1. Правильный ответ: С.
 - A. **Неправильно:** динамическое административное представление `sys.dm_exec_query_stats` предоставляет статистические данные о запросах, а не об индексах.
 - B. **Неправильно:** динамическое административное представление `sys.dm_exec_query_text` предоставляет текст пакетов и запросов.
 - C. **Правильно:** динамическое административное представление `sys.dm_db_index_usage_stats` предоставляет информацию об использовании индексов.
 - D. **Неправильно:** `sys.indexes` — это представление каталога, а не динамическое административное представление.
2. Правильный ответ: А.
 - A. **Правильно:** отсутствие необходимого количества данных является самым важным недостатком динамического управляющего объекта.
 - B. **Неправильно:** хотя некоторые запросы, использующие динамические управляющие объекты, становятся довольно сложными, с этим легко справиться, узнав больше о T-SQL и динамических управляющих объектах.
 - C. **Неправильно:** динамические управляющие объекты доступны во всех выпусках SQL Server.
 - D. **Неправильно:** динамические управляющие объекты — это системные объекты; вы не можете удалять или создавать их.
3. Правильный ответ: В.
 - A. **Неправильно:** динамическое административное представление `sys.dm_exec_query_stats` не предоставляет текст запроса.
 - B. **Правильно:** текст запроса можно получить, выполнив запрос к функции динамического управления `sys.dm_exec_sql_text`.
 - C. **Неправильно:** динамическое административное представление `sys.dm_exec_query_plan` не предоставляет текст запроса.
 - D. **Неправильно:** текст запроса можно получить, выполнив запрос к функции динамического управления `sys.dm_exec_sql_text`.

Упражнения

Упражнение 1. Анализ запросов

Вы получили срочный вызов от менеджера компании, в которой занимаетесь поддержкой SQL Server. Менеджер жалуется на то, что база данных SQL Server не отвечает уже несколько часов. Вам нужно оптимизировать только один запрос, но как можно быстрее. Но вы должны найти наиболее проблемный запрос. Вы подключились к экземпляру SQL Server. Вы видите, что сотни пользователей одновременно используют сервер, но при этом не запущены ни подсистема расширенных событий, ни трассировка SQL. Вы также выяснили, что SQL Server работает без перерыва уже 6 месяцев.

1. С чего вы начнете анализировать эту ситуацию?
2. Когда вы найдете наиболее проблемный запрос, что вы будете делать дальше?

Упражнение 2. Непрерывный мониторинг

Вам нужно постоянно выполнять мониторинг экземпляра SQL Server, для того чтобы выявлять потенциальные проблемные места. Ваш экземпляр SQL Server используется в интенсивном режиме. Вы не должны увеличивать нагрузку на него процедурами мониторинга.

1. Какой инструмент будете вы использовать для мониторинга?
2. Как вы минимизируете влияние этого инструмента?

Ответы

Упражнения

Упражнение 1. Анализ запросов

1. Вам нужно использовать связанные с выполнением динамические административные объекты для нахождения наиболее проблемных запросов.
2. Вы можете использовать графический предполагаемый план выполнения для этого запроса, чтобы найти операторы, имеющие наивысшую стоимость. Также

686

Ответы

вы можете проверить, имеются ли отсутствующие индексы, найденные связанными с индексом динамическими административными представлениями, которые могут быть полезны для того запроса.

Упражнение 2. Непрерывный мониторинг

1. Вам следует использовать подсистему расширенных событий SQL Server, как наиболее легкую систему мониторинга производительности.
2. В вашем случае вы должны выполнять мониторинг только нескольких самых важных событий. Вам следует собирать данные только для необходимых вам полей. Вы также должны отфильтровать события сеанса, чтобы включать только события, которые вам действительно нужны.

15 Реализация индексов и статистика

15.1 Реализация индексов

15.1.1 Кучи

Куча — это довольно простая структура. Данные в куче не имеют никакой логической организации. Куча — это просто набор страниц и экстентов.

SQL Server отслеживает, какие страницы и экстенты принадлежат объекту, с помощью специальных системных страниц, называемых страницами карты распределения индекса (Index Allocation Map, IAM). Каждая таблица или индекс имеет по крайней мере одну страницу IAM, называемую первой страницей IAM. Одна страница IAM может указывать примерно на 4 Гбайт пространства. Страницы IAM объекта организованы в двунаправленный список. SQL Server хранит указатели на первые страницы IAM в собственных внутренних системных таблицах.

Следующий запрос извлекает основную информацию о таблице dbo.TestStructure

```
1 CREATE TABLE dbo.TestStructure
2   ( id INT NOT NULL ,
3     filler1 CHAR(36) NOT NULL ,
4     filler2 CHAR(216) NOT NULL );
5
6   SELECT OBJECT_NAME(object_id) AS table_name , name AS index_name , type ,
7         type_desc
8   FROM sys.indexes
9 WHERE object_id = OBJECT_ID(N'dbo.TestStructure' , N'U');
```

Столбец type хранит значение 0 для куч, 1 для кластеризованных таблиц (индексов) и 2 для некластеризованных индексов.

Можно узнать, сколько страниц выделено под объект, из функции динамического управления sys.dm_db_index_physical_stats или с помощью системной процедуры dbo.sp_spaceused.

```
1   SELECT index_type_desc , page_count ,
2         record_count , avg_page_space_used_in_percent
3   FROM sys.dm_db_index_physical_stats
```

```
4      (DB_ID(N'tempdb'), OBJECT_ID(N'dbo.TestStructure'), NULL, NULL, ,
5      DETAILED);
EXEC dbo.sp_spaceused @objname = N'dbo.TestStructure', @updateusage =
true;
```

Обратите внимание на последний столбец первого запроса, avg_space_used_in_percent. Этот столбец показывает внутреннюю фрагментацию. Внутренняя фрагментация означает, что страницы не заполнены.

15.1.2 Кластеризованные индексы

ВАЖНО!

Кластеризованный индекс — это таблица

Когда вы создаете кластеризованный индекс, вы не копируете данные; напротив, вы реорганизуете таблицу. Кроме того, данные не отсортированы физически; если вам нужны упорядоченные выходные данные запроса, вы должны использовать предложение ORDER BY.

Основную информацию об индексе можно получить, запросив функцию динамического управления sys.dm_db_index_physical_stats. Следующий фрагмент кода будет использоваться в этой части занятия много раз, поэтому в дальнейшем будем называть его кодом «проверки выделения кластеризованного индекса».

```
1  SELECT index_type_desc, index_depth, index_level, page_count,
2      record_count, avg_page_space_used_in_percent
3  FROM sys.dm_db_index_physical_stats
4      (DB_ID(N'tempdb'), OBJECT_ID(N'dbo.TestStructure'),
5      NULL, NULL, 'DETAILED');
```

От фрагментации можно избавиться, если перестроить или реорганизовать индекс. Реорганизация индекса — процесс более медленный, но менее ресурсоемкий. В общем случае, следует выполнять реорганизацию индекса, если внешняя фрагментация менее 30%, и перестраивать его, если она больше 30%. Следующий код перестраивает индекс.

```
1  ALTER INDEX idx_cl_filler1 ON dbo.TestStructure REBUILD;
```

КОНТРОЛЬНЫЙ ВОПРОС

- Какой тип ключа кластеризации вы выберете для среды OLTP?

Ответ на контрольный вопрос

- Для среды OLTP наилучшим выбором может быть короткий, уникальный и последовательный ключ кластеризации.

15.1.3 Реализация некластеризованных индексов

Операция извлечения строки из кучи называется уточняющим запросом RID. Если запрос очень избирательный и ищет одну строку или только небольшое количество строк, тогда индексный поиск с помощью уточняющего запроса RID очень эффективен.

Если таблица организована как сбалансированное дерево, тогда указателем строк является ключ кластеризации. Это означает, что когда SQL Server ищет строку, он должен пройти все уровни на некластеризованном индексе, а затем также все уровни кластеризованного индекса. Эта операция называется поиском ключа.

В SQL Server 2012 существует способ хранения некластеризованных индексов. Кроме стандартного хранилища строк, SQL Server 2012 может сохранять данные индекса столбец за столбцом в так называемом индексе columnstore. Индексы columnstore могут многократно ускорить запросы хранилища данных, от 10 до 100 раз.

15.1.4 Реализация индексированных представлений

```
1 CREATE VIEW Sales.QuantityByCountry
2 WITH SCHEMABINDING
3 AS
4 SELECT O.shipcountry, SUM(OD.qty) AS total_ordered,
5 COUNT_BIG(*) AS number_of_rows
6 FROM Sales.OrderDetails AS OD
7 INNER JOIN Sales.Orders AS O
8 ON OD.orderid = O.orderid
9 GROUP BY O.shipcountry;
10 GO
11 CREATE UNIQUE CLUSTERED INDEX idx_cl_shipcountry
```

```
12 |     ON Sales.QuantityByCountry(shipcountry);  
13 | GO
```

Резюме занятия

- Можно сохранить таблицу как кучу или как сбалансированное дерево. Если таблица сохранена как сбалансированное дерево, она является кластеризованной; она также называется кластеризованным индексом.
- Можно создать некластеризованный индекс на куче или на кластеризованной таблице.
- Также можно индексировать представление.

Закрепление материала

1. Какие уровни может иметь индекс? (Выберите все подходящие варианты.)
 - А. Промежуточный уровень.
 - Б. Уровень кучи.
 - С. Корневой уровень.
 - Д. Конечный уровень.
2. Сколько кластеризованных индексов можно создать на таблице?
 - А. 999.
 - Б. 16.
 - С. 1.
 - Д. 900.
3. Что является указателем строк, когда таблица сохранена как сбалансированное дерево?
 - А. RID.
 - Б. Индексный ключ columnstore.
 - С. Ключ кластеризации.
 - Д. Таблица никогда не сохраняется как сбалансированное дерево.

Ответы

Занятие 1

Закрепление материала

1. Правильные ответы: А, С и D.
 - А. **Правильно:** индекс может иметь 0 и более промежуточных уровней.
 - В. **Неправильно:** куча — это отдельная структура, а не уровень индекса.
 - С. **Правильно:** каждый индекс имеет корневой уровень с одной корневой страницей.
 - Д. **Правильно:** самый нижний уровень индекса — это конечный уровень.
2. Правильный ответ: С.
 - А. **Неправильно:** можно создать 999 некластеризованных индексов на таблице.
 - Б. **Неправильно:** можно иметь до 16 столбцов в составном ключе.
 - С. **Правильно:** может быть только 1 кластеризованный индекс, поскольку эта сама таблица, построенная на сбалансированном дереве.
 - Д. **Неправильно:** размер столбцов в ключе не должен превышать 900 байт.
3. Правильный ответ: С.
 - А. **Неправильно:** RID используется для куч.
 - Б. **Неправильно:** столбцы в индексе columnstore не используются как указатели ключа.
 - С. **Правильно:** ключ кластеризации является указателем строки, когда таблица хранится как сбалансированное дерево.
 - Д. **Неправильно:** кластеризованная таблица хранится как сбалансированное дерево.

15.2 Использование аргументов поиска

15.2.1 Аргументы поиска

Чтобы написать подходящий аргумент поиска SARG, вы должны быть уверены, что столбец, имеющий индекс, появляется в предикате отдельно, а не как параметр функции. Аргумент поиска SARG должен принимать форму столбца `inclusive_operator <value>` или столбца `<value> inclusive_operator`. Имя столбца должно стоять отдельно на одной стороне выражения, а константа или вычисляемое значение — появляться на другой стороне. В качестве операторов могут использоваться операторы `=, >, <, =>, <=,` `BETWEEN` и `LIKE`. Но оператор `LIKE` можно использовать, только если подстановочные символы `%` или `_` не стоят в начале строковой переменной, с которой сравнивается столбец.

КОНТРОЛЬНЫЙ ВОПРОС

- Какие предложения запроса по вашему мнению поддерживаются индексом?

Ответ на контрольный вопрос

- Список предложений, которые предположительно поддерживаются индексом, включает, но не ограничен предложениями `WHERE, JOIN, GROUP BY` и `ORDER BY`.

Резюме занятия

- Разные части запроса могут поддерживаться индексами.
- Следует рассматривать возможность использования предложений запроса `WHERE, JOIN, GROUP BY, ORDER BY` и `SELECT` с надлежащими индексами.
- Следует писать подходящие аргументы поиска, не включая в выражения столбцы ключей индексов.

Закрепление материала

1. Как можно поддерживать предложение `SELECT` запроса с помощью некластеризованного индекса, который уже используется для предложения `WHERE`?
 - A. Нужно использовать `SELECT *`.
 - B. Можно модифицировать индекс, который уже используется, включив столбцы из списка `SELECT`, не являющиеся частью ключа.
 - C. Можно добавить псевдонимы столбцов.
 - D. Предложение `SELECT` нельзя поддерживать индексами.
2. Если нужна сортировка, где SQL Server сортирует данные?
 - A. В текущей базе данных.
 - B. В главной базе данных.
 - C. В базе данных `msdb`.
 - D. SQL Server сортирует данные в памяти или сбрасывает их в `tempdb`, если они не помещаются в памяти.
3. Вы создали индекс для поддержки предложения `WHERE` запроса. Но SQL Server не использует индекс. В чем заключаются возможные причины? (Выберите все подходящие варианты.)
 - A. По аргументам предиката не может выполняться поиск.
 - B. SQL Server не может использовать индекс для поддержки предложения `WHERE`.
 - C. Предикат является недостаточно выборочным.
 - D. Вы работаете с базой данных `tempdb`, и SQL Server не использует индексы в этой базе данных.

Ответы

Занятие 2

Закрепление материала

1. Правильный ответ: В.
 - A. **Неправильно:** использование инструкции `SELECT *` очень нежелательно и конечно не помогает SQL Server использовать индексы.
 - B. **Правильно:** вы можете модифицировать индекс, который уже используется, так, чтобы включить столбцы из списка `SELECT`, не являющиеся частью ключа.
 - C. **Неправильно:** добавление псевдонимов столбцов не влияет на использование индексов.
 - D. **Неправильно:** вы можете поддерживать предложение `SELECT` индексами.
2. Правильный ответ: D.
 - A. **Неправильно:** SQL Server сортирует данные в памяти или сбрасывает данные в tempdb, если они не помещаются в памяти.
 - B. **Неправильно:** SQL Server сортирует данные в памяти или сбрасывает данные в tempdb, если они не помещаются в памяти.
 - C. **Неправильно:** SQL Server сортирует данные в памяти или сбрасывает данные в tempdb, если они не помещаются в памяти.
 - D. **Правильно:** SQL Server сортирует данные в памяти или сбрасывает данные в tempdb, если они не помещаются в памяти.
3. Правильные ответы: А и С.
 - A. **Правильно:** SQL Server не использует индекс для поддержки предложения `WHERE`, если по аргументам предиката не может выполняться поиск.
 - B. **Неправильно:** SQL Server поддерживает предложение `WHERE` индексами.
 - C. **Правильно:** SQL Server может решить не использовать индекс для поддержки предложения `WHERE`, если запрос является недостаточно выборочным.
 - D. **Неправильно:** SQL Server рассматривает использование индексов в контексте базы данных tempdb точно так же, как в контексте любой другой базы данных.

15.3 Основные понятия статистики

15.3.1 Автоматически создаваемая статистика

Существуют три параметра баз данных, оказывающие влияние на автоматическое создание статистики.

- `AUTO_CREATE_STATISTICS`. Когда этот параметр включен, SQL Server создает статистику автоматически.

- AUTO_UPDATE_STATISTICS. Когда этот параметр включен, SQL Server может автоматически обновлять статистику при достаточном количестве изменений в базовых таблицах и индексах.
- AUTO_UPDATE_STATISTICS_ASYNC. Этот параметр определяет, использует ли SQL Server синхронное или асинхронное обновление статистики, во время оптимизации запроса.

Резюме занятия

- Оптимизатор запросов SQL Server использует статистику для определения количества элементов в запросе.
- Кроме предоставления права поддержки автоматической статистики SQL Server, можно также поддерживать статистику вручную.

Закрепление материала

Закрепление материала

Для проверки знания материала, изложенного в данном занятии, ответьте на следующие вопросы. Ответы на эти вопросы и пояснения, почему тот или иной ответ верен или неверен, можно найти в *приложении "Ответы" в конце книги*.

1. Как может SQL Server рассчитать мощность запроса?
 - SQL Server хранит информацию о мощности в конечных страницах индекса.
 - SQL Server быстро выполняет запрос на 10% пробных данных.
 - SQL Server не может рассчитать мощность запроса, если не представлена табличная подсказка.
 - SQL Server использует статистику, чтобы рассчитать мощность запроса.
2. Что из перечисленного ниже не является основанием для ручного обновления статистики?
 - Вы только что перестроили индекс.
 - Вы выполнили пакетную вставку большого количества данных в таблицу и хотите запросить эту таблицу сразу после вставки.
 - Вы обновили базу данных.
 - Запросы выполняются медленно, но вы знаете, что запросы написаны правильно и имеют надлежащие индексы.
3. Каково ограничение на количество шагов в статистических гистограммах?
 - 200 шагов на гистограмму.
 - 200 гистограмм на столбец.
 - 200 страниц на гистограмму.
 - 200 шагов на гистограмму.

Ответы

Закрепление материала

1. Правильный ответ: D.
 - A. **Неправильно:** информации о мощности нет на страницах конечного уровня или в индексах.
 - B. **Неправильно:** SQL Server не выполняет запрос заранее на данных образца.
 - C. **Неправильно:** SQL Server может предварительно рассчитать мощность запроса.
 - D. **Правильно:** SQL Server использует статистику для предварительного расчета мощности запроса.

688

Ответы

2. Правильный ответ: A.
 - A. **Правильно:** при перестройке индекса SQL Server автоматически обновляет статистику.
 - B. **Неправильно:** после пакетной вставки большого количества данных в таблицу, если вы хотите выполнить запрос к ней сразу после этого, нужно обновить статистику для этой таблицы.
 - C. **Неправильно:** нужно обновить статистику для всей базы данных после ее обновления.
 - D. **Неправильно:** нужно обновить статистику, если запросы выполняются медленно и вы знаете, что они написаны правильно и поддерживаются надлежащими индексами.
3. Правильный ответ: D.
 - A. **Неправильно:** в гистограмме можно иметь до 200 шагов.
 - B. **Неправильно:** можно иметь 1 гистограмму на статистику.
 - C. **Неправильно:** существует ограничение количества шагов на гистограмму.
 - D. **Правильно:** в гистограмме можно иметь до 200 шагов.

Упражнения

Упражнение 1. Просмотр таблицы

Администраторы баз данных в компании, в которой вы занимаетесь поддержкой баз данных SQL Server, жалуются, что для большинства запросов SQL Server про-

сматривает таблицы целиком, хотя запросы являются выборочными. Производительность недопустимо низкая. Вы должны помочь повысить производительность.

1. Какие физические структуры вам нужно проверить?
2. Нужно ли вам также проверять код?

Упражнение 2. Медленные обновления

Конечные пользователи в компании, где вы отвечаете за оптимизацию баз данных, жалуются на то, что обновления баз данных происходят очень медленно, даже если обновляется всего 1 строка. Поиск обновленных строк поддерживается соответствующими индексами. Запросы SELECT выполняются хорошо. Это именно то, чего вы ожидали, поскольку вы создали некластеризованные индексы на всех столбцах, используемых в этих запросах. Вы должны также повысить производительность базы данных для обновлений.

1. Как вы думаете, что является причиной медленных обновлений?
2. Как вы будете исследовать возможные проблемы?

Ответы

Упражнения

Упражнение 1. Просмотр таблицы

1. Вам нужно проверить, поддерживаются ли запросы индексами. Кроме того, вам следует проверить, создается и поддерживается ли статистика для индексов.
2. Вам надо проверить, используют ли запросы надлежащие аргументы поиска.

Упражнение 2. Медленные обновления

1. Слишком большое число индексов может замедлить обновления. Вероятно, вы создали множество ненужных индексов, но поскольку SQL Server должен их поддерживать, обновления выполняются медленно.
2. Вы можете запросить динамическое административное представление `sys.dm_db_index_usage_stats`, чтобы определить, какие индексы используются для поиска, а какие — только для обновления.

16 Основные сведения о курсорах, наборах данных и временных таблицах

16.1 Сравнительная оценка использования решений на основе курсоров/итераций и решений на основе наборов данных

Контрольные вопросы

1. Какие команды требуются для работы с курсором?
2. Что означает использование параметра `FAST_FORWARD` в команде объявления курсора с точки зрения свойств курсора?

Ответы на контрольные вопросы

1. `DECLARE, OPEN, FETCH в цикле, CLOSE и DEALLOCATE.`
2. `Что этот курсор является односторонним и доступным только для чтения.`

Резюме занятия

- Вы можете использовать один из двух основных подходов к решению связанных с запросами задач; один — использование решения на основе наборов данных, другой — использование итерационных решений.
- Решения на основе наборов используют SQL-запросы, которые следуют принципам реляционной модели. Они взаимодействуют с входными таблицами (наборами) как с единым целым, в отличие от построчного взаимодействия. Они также не ожидают, что данные будут приниматься или возвращаться в определенном порядке.
- Некоторые задачи должны решаться с помощью итерационных решений, к ним относятся задачи управления, которые должны выполняться для каждого объекта, или хранимые процедуры, которые нужно выполнять в таблице построчно.

- Что касается связанных с запросами задач, обычно по умолчанию рекомендуется использовать решения на основе наборов, а итерационные решения применять только в исключительных случаях.

Закрепление материала

1. При извлечении строки из курсора как можно определить, что больше нет строк для извлечения?
 - А. Когда функция `@@FETCH_STATUS` возвращает 0.
 - Б. Когда функция `@@FETCH_STATUS` возвращает -1.
 - С. Когда функция `@@FETCH_STATUS` возвращает -2.
 - Д. Когда функция `@@FETCH_STATUS` генерирует ошибку.
2. Почему важно предпочитать решения на основе наборов итерационным решениям для связанных с запросами задач? (Выберите все подходящие варианты.)
 - А. Потому что решения на основе наборов основываются на реляционной модели, которая положена в основу языка T-SQL.

584

Глава 16

- Б. Потому что решения на основе наборов всегда обеспечивают более высокую производительность, чем итерационные решения.
 - С. Потому что решения на основе наборов обычно требуют написания меньшего количества кода, чем итерационные решения.
 - Д. Потому что решения на основе наборов дают возможность опереться на порядок данных.
3. Когда вам нужно обрабатывать одну строку за один раз, что вы можете использовать в качестве альтернативы курсора?
 - А. Использовать конструкцию цикла `FOR EACH`.
 - Б. Извлечь минимальный и максимальный ключи, а затем выполнить цикл со счетчиком, который начинает с минимального значения и увеличивается на 1 в каждой итерации, пока не достигнет максимума.
 - С. Использовать запрос `TOP (1)`, упорядоченный по ключу, для выбора первой строки. Затем использовать цикл, пока последний возвращенный ключ не равен `NULL`. В каждой итерации цикла обрабатывать текущую строку и затем использовать запрос `TOP (1)` с ключом, большим последнего значения, отсортированный по этому ключу, для выборки следующей строки.
 - Д. Определить построчный триггер `SELECT`.

Ответы

1. Правильный ответ: В.
 - A. **Неправильно:** 0 означает, что последняя выборка была успешной. Могут быть другие строки для выборки.
 - B. **Правильно:** -1 означает, что строка находится вне результирующего набора.

- C. **Неправильно:** -2 означает, что выбранная строка отсутствует. Также могут быть другие строки для выборки.
 - D. **Неправильно:** функция не должна генерировать сообщение об ошибке.
2. Правильные ответы: А и С.
 - A. **Правильно:** решения на основе наборов базируются на принципах реляционной модели, которая положена в основу языков SQL (стандартный язык) и T-SQL (диалект языка в SQL Server).
 - B. **Неправильно:** хотя это исключение из правила, но иногда итерационные решения могут быть более быстрыми, чем решения на основе наборов.
 - C. **Правильно:** поскольку решения на основе наборов являются декларативными, а итерационные — императивными, решения на основе наборов имеют тенденцию использовать меньшее количество кода.
 - D. **Неправильно:** решения на основе наборов не могут делать никаких заключений о последовательности данных, поскольку наборы данных являются неупорядоченными.
 3. Правильный ответ: С.
 - A. **Неправильно:** T-SQL не поддерживает цикл FOR EACH.
 - B. **Неправильно:** при наличии разрывов между ключами этот подход приведет к попытке обрабатывать несуществующие ключи.
 - C. **Правильно:** подход с использованием параметра TOP действительно является хорошей альтернативой курсора. Но следует принимать во внимание, что он вызывает увеличение нагрузки на ввод-вывод.
 - D. **Неправильно:** в языке T-SQL не существует триггера SELECT или построчных триггеров.

16.2 Сравнение использования временных таблиц и табличных переменных

Табличные переменные хорошо использовать в двух общих случаях. Один — когда объем данных настолько маленький, как страница или две, что эффективность плана не имеет значения. Другой случай — когда план очень простой. Простой план означает, что существует только один разумный план, и оптимизатору фактически не нужны гистограммы для принятия решения.

Контрольные вопросы

1. Как вы создадите локальную временную таблицу и как — глобальную?
2. Можно ли присваивать имена ограничениям в локальных временных таблицах и в табличных переменных?

Ответы на контрольные вопросы

1. Нужно присвоить имя локальной временной таблице, используя один знак решетки # в качестве префикса, а глобальной локальной переменной — с использованием двух знаков решетки в качестве префикса.
2. Можно присваивать имена ограничениям в локальных временных таблицах, хотя это не рекомендуется делать, поскольку это может привести к конфликту имен. Присваивать имена ограничениям в табличных переменных нельзя.

Резюме занятия

- Вы можете использовать временные таблицы и табличные переменные, когда вам нужно временно сохранить данные, такие как промежуточный результирующий набор запроса.
- Временные таблицы и табличные переменные имеют множество различий, включая область действия, язык DDL и индексирование, взаимодействие с транзакциями и статистику распространения.
- Локальные временные таблицы видны уровню, который их создал, всем пакетам этого уровня, а также внутренним и уровням в стеке вызовов. Табличные переменные видны только пакету, который их объявил.
- Можно применять язык DDL к временной таблице после ее создания, включая создание индексов и другие изменения DDL. Нельзя приме-

нять изменения DDL к табличной переменной после ее объявления. В табличной переменной можно создавать индексы косвенным образом посредством ограничений первичного ключа и уникальности.

- Изменения, примененные к временной таблице в транзакции, отменяются при откате этой транзакции. Изменения, примененные к табличной переменной, не отменяются при откате транзакции.
- SQL Server поддерживает статистику распространения на временных таблицах, но не на табличных переменных. В результате, планы запросов, использующих временные таблицы, более оптимальны, чем планы с использованием табличных переменных.

Закрепление материала

1. Какой из перечисленных вариантов подходит для использования табличных переменных? (Выберите все подходящие варианты.)
 - А. Когда таблицы очень маленькие, а план — простой.
 - В. Когда таблицы очень маленькие, а план — непростой.
 - С. Когда таблицы большие, а план — простой.
 - Д. Когда таблицы большие, а план — непростой.
2. Можно ли создать индексы на табличных переменных?
 - А. Нет.
 - В. Да, выполнив команду `CREATE INDEX`.
 - С. Да, косвенно, определив первичный ключ и ограничения уникальности.
 - Д. Да, с помощью определения внешнего ключа.
3. Перед вами поставлена задача реализовать триггер. Как часть кода триггера в особых условиях, вам нужно выполнять откат транзакции. Но вам требуется скопировать данные из вставленных и удаленных таблиц в таблицы аудита, чтобы отслеживать изменения. Как вы можете сделать это?
 - А. Выполнить откат транзакции и затем скопировать данные из удаленных и вставленных таблиц в таблицы аудита.
 - В. Скопировать данные из вставленных и удаленных таблиц в таблицы аудита и затем выполнить откат транзакции.
 - С. Скопировать строки из вставленных и удаленных таблиц во временные таблицы, выполнить откат транзакции и затем скопировать данные из временных таблиц в таблицы аудита.
 - Д. Скопировать строки из вставленных и удаленных таблиц в табличные переменные, выполнить откат транзакции и затем скопировать данные из табличных переменных в таблицы аудита.

Ответы

1. Правильные ответы: А, В и С.
 - A. **Правильно:** табличные переменные пригодны для использования, когда таблицы очень маленькие.
 - B. **Правильно:** табличные переменные пригодны для использования, когда таблицы очень маленькие.
 - C. **Правильно:** при простом плане табличные переменные по-прежнему пригодны для использования, даже если они большие.
 - D. **Неправильно:** если таблицы большие и план непростой, предпочтительными являются временные таблицы.
2. Правильный ответ: С.
 - A. **Неправильно:** на табличных переменных могут существовать индексы.
 - B. **Неправильно:** команда CREATE INDEX не поддерживается для табличных переменных.

690

Ответы

- C. **Правильно:** можно создать индексы косвенным образом, определив ограничения первичного ключа и уникальности.
 - D. **Неправильно:** внешние ключи не создают индексы; более того, они не поддерживаются на временных таблицах и табличных переменных.
3. Правильный ответ: D.
 - A. **Неправильно:** после отката транзакции в триггере вставленные и удаленные таблицы очищаются.
 - B. **Неправильно:** откат транзакции вызывает отмену копирования в таблицы аудита.
 - C. **Неправильно:** изменения во временных таблицах отменяются после отката транзакции.
 - D. **Правильно:** изменения в табличных переменных не отменяются после отката транзакции, поэтому такое решение является правильным.

Упражнения

Упражнение 1. Рекомендации по повышению производительности для курсоров и временных объектов

Вы наняты в качестве консультанта в новую компанию, которая разрабатывает приложение, использующее SQL Server в качестве базы данных. Компания в настоящее время сталкивается с проблемами производительности и масштабируемости. Вы изучили код и выяснили следующее.

Почти все решения выполнены как курсоры. Изучив эти решения, вы увидели, что это не те случаи, которые должны быть реализованы с помощью итерационной логики.

Некоторые решения хранят промежуточные результаты в табличных переменных и затем запрашивают эти табличные переменные. Большое количество строк хранится в табличных переменных.

1. Можете ли вы дать рекомендации по поводу факта использования большинством решений курсоров?
2. Можете ли вы дать рекомендации по поводу использования табличных переменных?
3. Можете ли вы объяснить клиенту, при каких обстоятельствах должны использоваться курсоры и табличные переменные?

Упражнение 2. Указать неточности в ответах

Вы участвуете в конференции и присутствуете на лекции, посвященной языку T-SQL. В конце конференции лектор предлагает задать вопросы и отвечает на них. Далее приведены вопросы, которые слушатели задавали лектору, и ответы лектора на них. Укажите неточности в ответах лектора.

1. Вопрос: в чем заключается разница между временными таблицами и табличными переменными с точки зрения производительности?

Ответ: никакой разницы. Компания Microsoft просто предлагает немного разные способы решения одной и той же задачи.

2. Вопрос: у меня есть многострочный триггер UPDATE, который устанавливает значение столбца lastmod в измененных строках в значение, возвращенное функцией SYSDATETIME(). Этот триггер применяет курсор к вставленным таблицам для построчной обработки. Триггер работает очень медленно. Можете дать рекомендации по увеличению производительности триггера?

Ответ: вместо использования курсора напишите решение на основе наборов данных, которое использует цикл WHERE и запрос TOP для выполнения итераций по ключам по одному за один раз.

3. Вопрос: приведите, пожалуйста, пример, когда полезны табличные выражения?

Ответ: один из примеров — ситуация, когда требуется сохранить результат ресурсоемкого запроса и затем многократно ссылаться на этот результат.

Ответы

Упражнение 1. Рекомендации по повышению производительности для курсоров и временных объектов

1. Клиент должен оценить использование решений на основе наборов вместо решений на основе курсоров. Если большинство существующих решений использует курсоры, возможно, проблема заключается в недостатке понимания концепций реляционной модели. Можно порекомендовать компании организовать курс обучения для разработчиков по этой теме.
2. Когда большое количество строк должно быть сохранено во временных объектах, возможность оптимизатора выполнить точный прогноз выборочности становится более важным для эффективности плана. Исключение — если план простой. SQL Server не поддерживает статистику распределения (гистограммы) на табличных переменных, поэтому прогноз может быть неточным. Неточный прогноз может привести к неоптимальному плану. Клиент должен проверить планы запросов и найти неточные прогнозы. И если они будут найдены, необходимо оценить, стоит ли в этих случаях использовать временные таблицы вместо табличных переменных. SQL Server не поддерживает гистограммы для временных таблиц, и поэтому план выполнения для них более оптимальный.
3. В некоторых случаях уместно использовать табличные переменные, например, если объем данных очень мал — страница или две. Также если таблица большая, а план простой, оптимизатору не нужны гистограммы для выбора оптимального плана. То, что табличные переменные не имеют гистограмм, дает некоторые преимущества. Вы избежите затрат на их поддержку. Также не будет затрат на повторную компиляцию планов выполнения, связанных с обновлениями гистограмм.

Что касается курсоров, в некоторых случаях необходимо выполнять обработку каждой отдельной строки в таблице. Например, в целях обслуживания может

понадобиться выполнять некоторую работу для каждого индекса, таблицы, базы данных или другого объекта. Для таких случаев предназначены курсоры. Что касается манипулирования данными, возможны случаи, когда оптимизатор запросов SQL Server не может выполнить оптимизацию запроса хорошо, и вы не можете найти способ помочь оптимизатору сгенерировать эффективный план. С помощью курсоров, несмотря на большие издержки, иногда можно получить лучшие результаты, поскольку у вас больше возможностей управления процессом. Но такие случаи являются исключениями из правил.

Упражнение 2. Указать неточности в ответах

1. Между временными таблицами и табличными переменными существуют различия с точки зрения производительности. Одно важное различие — это поддержка в SQL Server статистики распространения (гистограмм) применительно к временным таблицам, но не к табличным переменным. Это означает, что для временных таблиц оптимизатор может сделать лучший прогноз выборочности. Поэтому планы, включающие временные таблицы, более оптимальны по сравнению с планами, включающими табличные переменные.
2. Решение с использованием циклов с запросом `TOP` вместо курсора не будет основано на наборах и не всегда более эффективно, чем существующее решение с использованием курсора. Вы по-прежнему обрабатываете строки по одной за один раз. Лучшим решением может быть применение одной инструкции `UPDATE` или `MERGE`, которая соединяет вставленную таблицу с базовой таблицей и обновляет все целевые строки с помощью одной операции на основе набора.
3. Результат внутреннего запроса табличного выражения не сохраняется в рабочей таблице. SQL Server раскрывает все ссылки на табличные выражения и взаимодействует с базовыми объектами напрямую. Многократные ссылки раскрываются несколько раз, поэтому работа повторяется. Если требуется сохранить результат ресурсоемкого запроса, чтобы избежать повторной работы, следует рассмотреть возможность использования временных таблиц или табличных переменных. Например, полезно использовать табличные выражения, когда необходимо ссылааться на псевдонимы столбцов, сгенерированные в списке `SELECT` в предложениях, которые логически обрабатываются раньше инструкции `SELECT`, таких как `WHERE`, `GROUP BY` и `HAVING`.

17 Основные сведения о дальнейших аспектах оптимизации

17.1 Общие сведения об итераторах планов

Резюме занятия

- SQL Server использует много различных методов доступа к данным.
- SQL Server использует разные алгоритмы соединений и статистической обработки.
- Не существует "хороших" и "плохих" итераторов. Любой итератор может подходить наилучшим образом для конкретного запроса и конкретных данных.

Закрепление материала

Закрепление материала

Для проверки знания материала, изложенного в данном занятии, ответьте на следующие вопросы. Ответы на эти вопросы и пояснения, почему тот или иной ответ верен или неверен, можно найти в приложении "Ответы" в конце книги.

1. Какие алгоритмы статистической обработки использует SQL Server? (Выберите все подходящие варианты.)
 - A. Статистическая обработка слияний.
 - B. Статистическая обработка потока.
 - C. Статистическая обработка хэша.
 - D. Статистическая обработка вложенных циклов.
2. Какой оператор используется, когда SQL Server выполняет поиск по некластеризованному индексу для нахождения строки, но также ему нужны данные из базовой таблицы, организованной как кластеризованный индекс?
 - A. Уточняющий запрос RID.
 - B. Просмотр кластеризованного индекса.

- C. Соединение слиянием.
- D. Уточняющий запрос по ключу.
3. Как называется просмотр, когда SQL Server просматривает кластеризованный индекс в логической последовательности индекса?
 - A. Просмотр в порядке выделения.
 - B. Просмотр по кластеризованному индексу.
 - C. Просмотр в порядке индекса.
 - D. Поиск в порядке индекса.

Ответы

Занятие 1

Закрепление материала

1. Правильные ответы: В и С.
 - A. **Неправильно:** не существует алгоритма статистической обработки слияния.
 - B. **Правильно:** SQL Server использует алгоритмы статистической обработки потока и хэша.

692

Ответы

- C. **Правильно:** SQL Server использует алгоритмы статистической обработки потока и хэша.
 - D. **Неправильно:** не существует алгоритма статистической обработки вложенных циклов.
2. Правильный ответ: D.
 - A. **Неправильно:** SQL Server использует оператор **RID Lookup** (Уточняющий запрос RID) для получения данных из базовой таблицы после поиска по некластеризованному индексу, если базовая таблица организована как куча.
 - B. **Неправильно:** нет необходимости в выполнении полного просмотра кластеризованного индекса после того, как строка найдена в некластеризованном индексе.
 - C. **Неправильно:** оператор **Merge Join** (Соединение слиянием) используется при выполнении соединения слиянием.
 - D. **Правильно:** SQL Server использует оператор **Key Lookup** (Уточняющий запрос ключа) для получения данных из базовой кластеризованной таблицы после поиска по некластеризованному индексу.
 3. Правильный ответ: C.
 - A. **Неправильно:** в физической последовательности этот просмотр называется просмотром в порядке выделения.
 - B. **Неправильно:** **Clustered Index Scan** (Просмотр кластеризованного индекса) — это оператор, который может просматривать данные в физической или логической последовательности.
 - C. **Правильно:** когда SQL Server просматривает данные в логическом порядке индекса, такой просмотр называется просмотром в порядке индекса.
 - D. **Неправильно:** оператор **Seek** (Поиск) не используется для просмотра.

17.2 Использование параметризованных запросов и пакетных операций

Резюме занятия

- SQL Server выполняет параметризацию запросов для улучшения повторного использования планов выполнения.
- Вы можете выполнить параметризацию запросов самостоятельно.
- Режим пакетной обработки, который появился только в SQL Server 2012, может значительно повысить производительность запросов к хранилищам данных, особенно снижением использования ЦП.
- Пакетная обработка хорошо совмещается с индексами columnstore.

Закрепление материала

1. Укажите возможные причины, по которым SQL Server не использует повторно существующий кэшированный план? (Выберите все подходящие варианты.)
 - A. Потому что параметр SET, который влияет на результат запроса, был изменен.
 - B. Потому что запрос был параметризован вручную в хранимой процедуре, и не используется параметр повторной компиляции.
 - C. Потому что SQL Server не может определить выборочность параметризованного предиката.
 - D. Потому что в параметризованном предикате для параметра использовался другой тип данных.
2. Каково главное преимущество пакетного режима обработки?
 - A. Он снижает дисковый ввод-вывод.
 - B. Он ускоряет передачу по сети.
 - C. Он снижает использование ЦП.
 - D. Он использует меньше памяти.
3. Какая из следующих команд SET препятствует повторному использованию плана выполнения?
 - A. SET STATISTICS IO ON.
 - B. SET STATISTICS PROFILE ON.
 - C. SET CONCAT_NULL_YIELDS_NULL OFF.
 - D. SET STATISTICS IO OFF.

Ответы

Занятие 2

Закрепление материала

1. Правильные ответы: А, С и D.
 - A. **Правильно:** изменение параметра `SET` может быть причиной того, что SQL Server не использует повторно существующий кэшированный план.
 - B. **Неправильно:** SQL Server повторно использует кэшированные планы хранимой процедуры, если вы не запускаете повторную компиляцию или используете в процедуре динамический SQL.
 - C. **Правильно:** SQL Server не использует повторно кэшированный план, если он не может определить применяемость параметризованного предиката.
 - D. **Правильно:** если в параметризованном предикате используется другой тип данных для параметра, SQL Server повторно не использует кэшированный план.

Ответы

693

2. Правильный ответ: С.
 - A. **Неправильно:** обработка в пакетном режиме не уменьшает дисковый ввод-вывод.
 - B. **Неправильно:** обработка в пакетном режиме не оказывает влияния на сеть.
 - C. **Правильно:** снижение использования ЦП — главное преимущество обработки в пакетном режиме.
 - D. **Неправильно:** использование памяти не связано напрямую с пакетной обработкой.
3. Правильный ответ: С.
 - A. **Неправильно:** команда `SET STATISTICS IO` только включает и отключает статистическую информацию о дисковом вводе-выводе.
 - B. **Неправильно:** когда параметр `STATISTICS PROFILE` установлен в `ON`, каждый выполненный запрос возвращает свой обычный результирующий набор, стоящий перед дополнительным результирующим набором, который показывает профиль выполнения запроса; он не влияет на повторную используемость плана выполнения.
 - C. **Правильно:** команда `SET CONCAT_NULL_YIELDS_NULL` влияет на результат запроса и, следовательно, ее изменение не позволяет повторно использовать план.
 - D. **Неправильно:** команда `SET STATISTICS IO` только включает и отключает статистическую информацию о дисковом вводе-выводе.

Занятие 3

Упражнения

Упражнение 1. Оптимизация запроса

Администраторы базы данных в компании, где вы занимаетесь поддержкой SQL Server, жалуются, что SQL Server не использует индексы для некоторых запросов от стороннего приложения. Запросы генерированы непосредственно в приложении, поэтому вы не можете их изменить.

1. Какие действия вы можете предпринять?
2. Можно ли оптимизировать запросы, которые вы не можете модифицировать?

Упражнение 2. Табличная подсказка

В хранимой процедуре есть запрос, для которого, как вам кажется, SQL Server не использует оптимальный план. Хотя очень избирательное предложение `WHERE` запроса поддерживается некластеризованным индексом, SQL Server не использует его. Статистика для всех индексов в актуальном состоянии. Вам нужно оптимизировать этот запрос.

1. Можете ли вы модифицировать запрос, чтобы использовать индекс?
2. Как вы выполните эту задачу?

Ответы

Упражнения

Упражнение 1. Оптимизация запроса

1. Можно проверить, являются ли индексы наиболее подходящими для запросов. Также вы можете проверить, обновлены ли статистические данные для индексов. Кроме того, вы можете создать структуры планов для проблематичных запросов.
2. Нет, вы не можете использовать подсказки оптимизатора, поскольку не можете изменить текст проблематичных запросов.

Упражнение 2. Табличная подсказка

1. Да. Поскольку запрос находится внутри хранимой процедуры и не встроен в приложение, вы можете изменить его.
2. Вы можете использовать табличную подсказку, чтобы инициировать использование индекса.