



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	5
1.1 Основы языка T-SQL	5
1.2 Понимание логической обработки запросов	8
2 Начало работы с инструкцией SELECT	14
2.1 Использование FROM и SELECT	14
2.2 Работа с типами данных и встроенными функциями	17
2.2.1 Выбор типов данных для ключей	18
2.2.2 Выражение CASE и связанные с ним функции	20
2.2.3 Функция COALESCE	20
3 Фильтрация и сортировка данных	28
3.1 Фильтрация данных с помощью предикатов	28
3.1.1 Предикат LIKE	29
3.1.2 Фильтрация данных даты и времени	30
3.2 Сортировка данных	34
3.3 Фильтрация данных с помощью TOP и OFFSET...FETCH . . .	38
3.3.1 Фильтрация данных с помощью предложения TOP . . .	38
3.3.2 Фильтрация данных с помощью OFFSET...FETCH . . .	39
4 Комбинирование наборов данных	49
4.1 Использование соединений	49
4.1.1 Перекрестные соединения CROSS JOIN	49
4.1.2 Внутренние соединения INNER JOIN	50
4.1.3 Внешние соединения OUTER JOIN	51
4.2 Использование подзапросов, табличных выражений и оператора APPLY	56
4.2.1 Независимые подзапросы	56
4.2.2 Коррелированные (связанные) подзапросы	56
4.3 Табличные выражения	57
4.3.1 Производные таблицы	58
4.3.2 Обобщенные табличные выражения	59
4.3.3 Представления и встроенные табличные функции	61

4.4	Оператор APPLY	63
4.4.1	CROSS APPLY	63
4.4.2	OUTER APPLY	64
4.5	Использование операторов работы с наборами	70
4.5.1	Операторы UNION и UNION ALL	71
4.5.2	Оператор INTERSECT	73
4.5.3	Оператор EXCEPT	73
5	Группирование и оконные функции	79
5.1	Написание запросов для группировки данных	79
5.2	Работа с несколькими наборами группирования	79
5.2.1	GROUPING SETS	80
5.2.2	CUBE	81
5.2.3	ROLLUP	81
5.2.4	GROUPING, GROUPING_ID	83
5.3	Сведение и отмена сведения данных	88
5.3.1	Сведение данных	88
5.3.2	Отмена сведения данных	89
5.4	Использование оконных функций	93
5.4.1	Статистические оконные функции	93
5.4.2	Ранжирующие оконные функции	94
5.4.3	Оконные функции смещения	96
6	Запросы с полнотекстовым поиском данных	103
6.1	Создание полнотекстовых каталогов и индексов	103
6.1.1	Компоненты полнотекстового поиска	103
6.2	Использование предикатов CONTAINS и FREETEXT	106
6.2.1	Предикат CONTAINS	106
6.2.2	Предикат FREETEXT	106
6.3	Использование табличных функций полнотекстового и семантического поиска	107
6.3.1	Статистические оконные функции	107
7	Запрос и управление XML-данными	108

7.1	Возвращение результатов в виде XML с помощью предложения FOR XML	108
7.1.1	Режим FOR XML PATH	109
7.2	Запрос XML-данных с помощью XQuery	109
8	Создание таблиц и обеспечение целостности данных	111
8.1	Создание и изменение таблиц	111
8.1.1	Создание таблицы	111
8.1.2	Определение схемы базы данных	111
8.1.3	Свойство идентификатора и порядковые номера	114
8.1.4	Сжатие таблиц	114
8.1.5	Изменение таблицы	115
8.2	Обеспечение целостности данных	118
8.2.1	Использование ограничений	118

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.

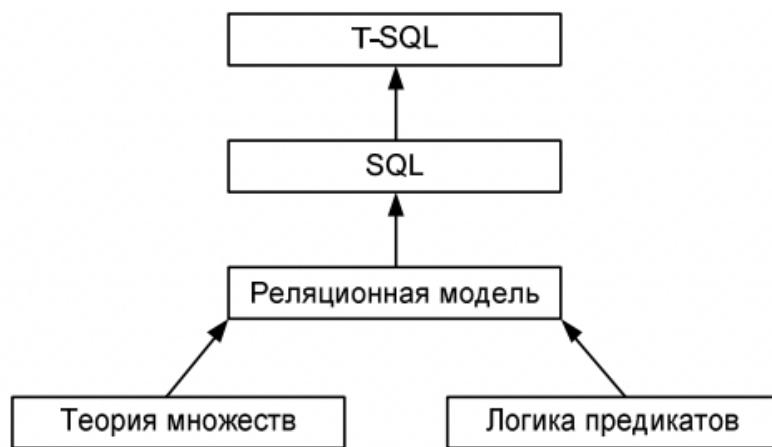


Рис. 1.1: Эволюция языка T-SQL

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. Ссыльайтесь 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.

- 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

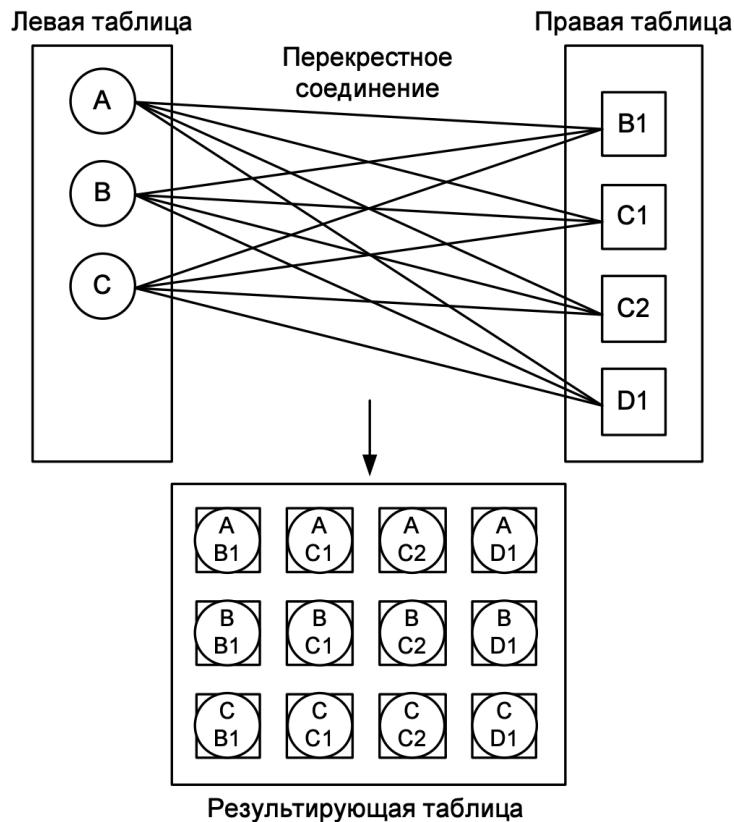


Рис. 4.1: Перекрестное объединение таблиц

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

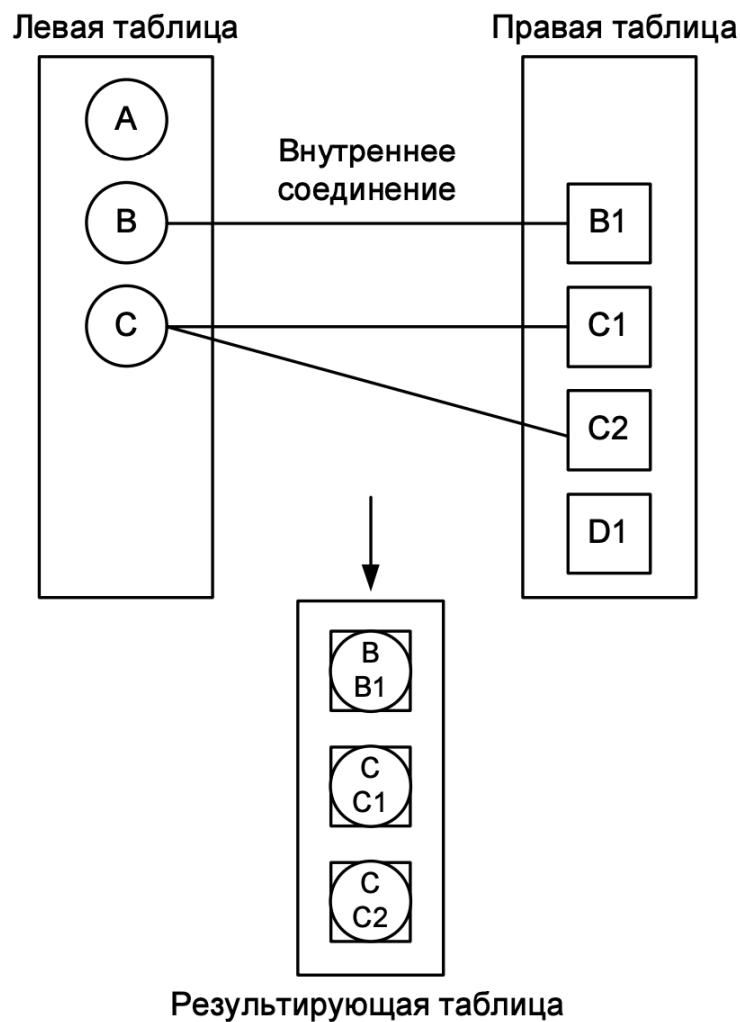


Рис. 4.2: Внутреннее объединение таблиц

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.

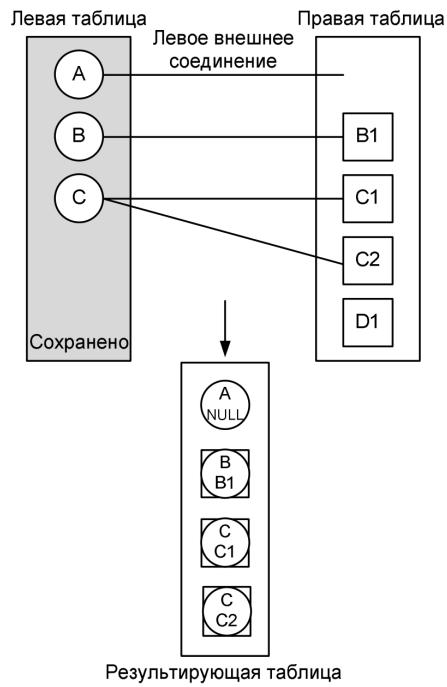


Рис. 4.3: Левое внешнее соединение

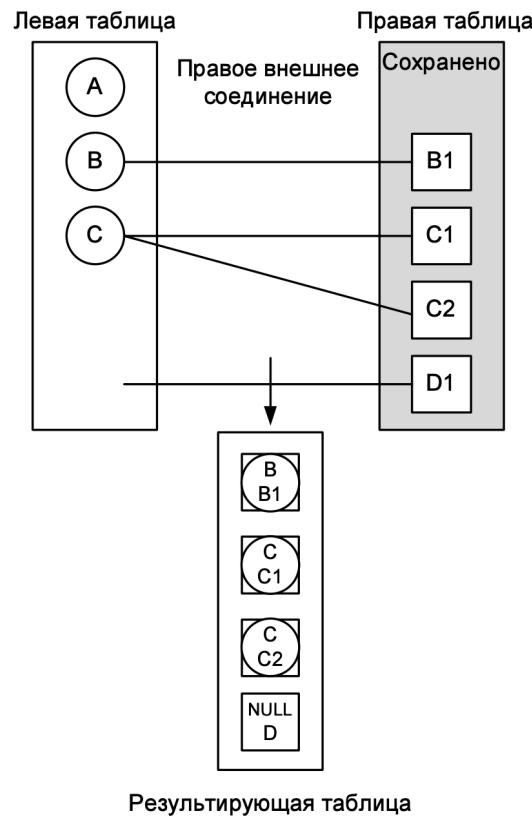


Рис. 4.4: Правое внешнее соединение

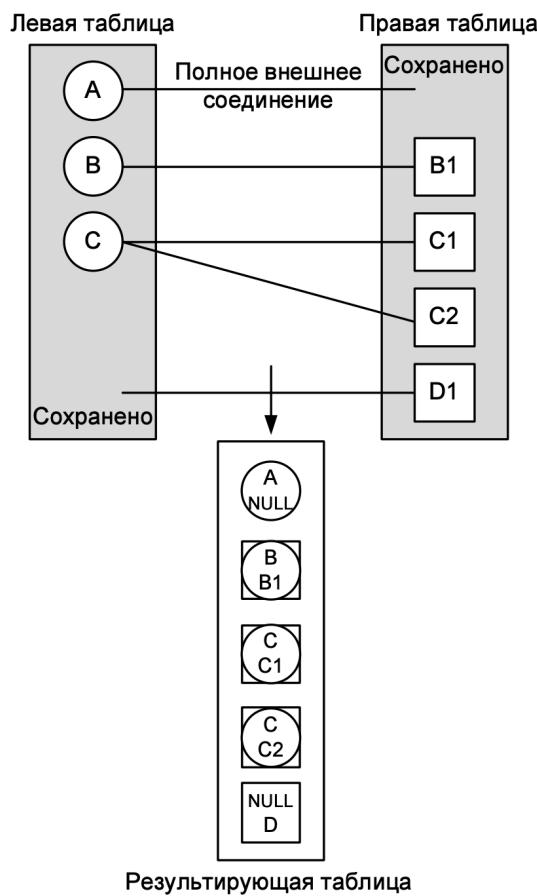


Рис. 4.5: Полное внешнее соединение

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:

```

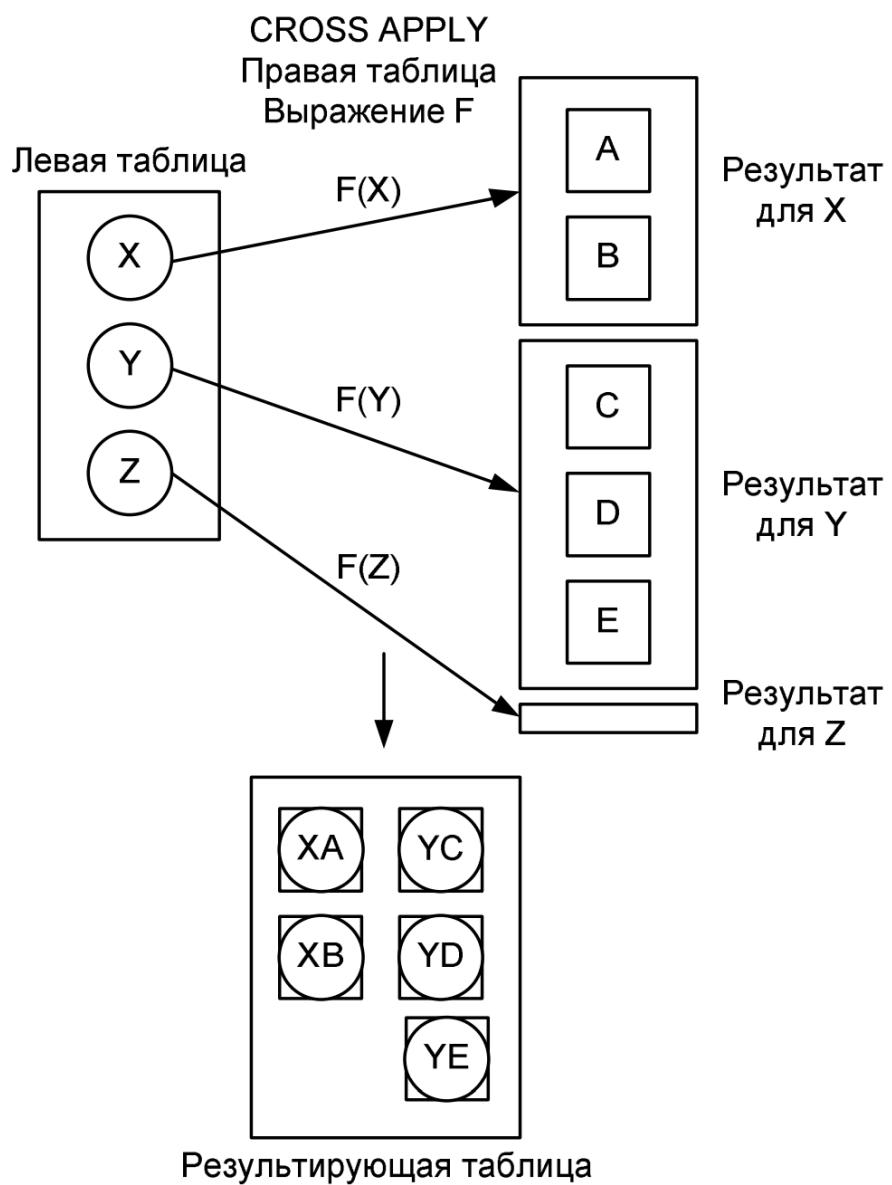
1 IF OBJECT_ID('HR.GetManagers', 'IF') IS NOT NULL DROP FUNCTION HR.
2   GetManagers;
3 GO
4 CREATE FUNCTION HR.GetManagers(@empid AS INT) RETURNS TABLE
5 AS
6 RETURN
7 WITH EmpsCTE AS
8 ( SELECT empid, mgrid, firstname, lastname, 0 AS distance
9   FROM HR.Employees
10  WHERE empid = @empid
11 UNION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

```

4.4 Оператор APPLY

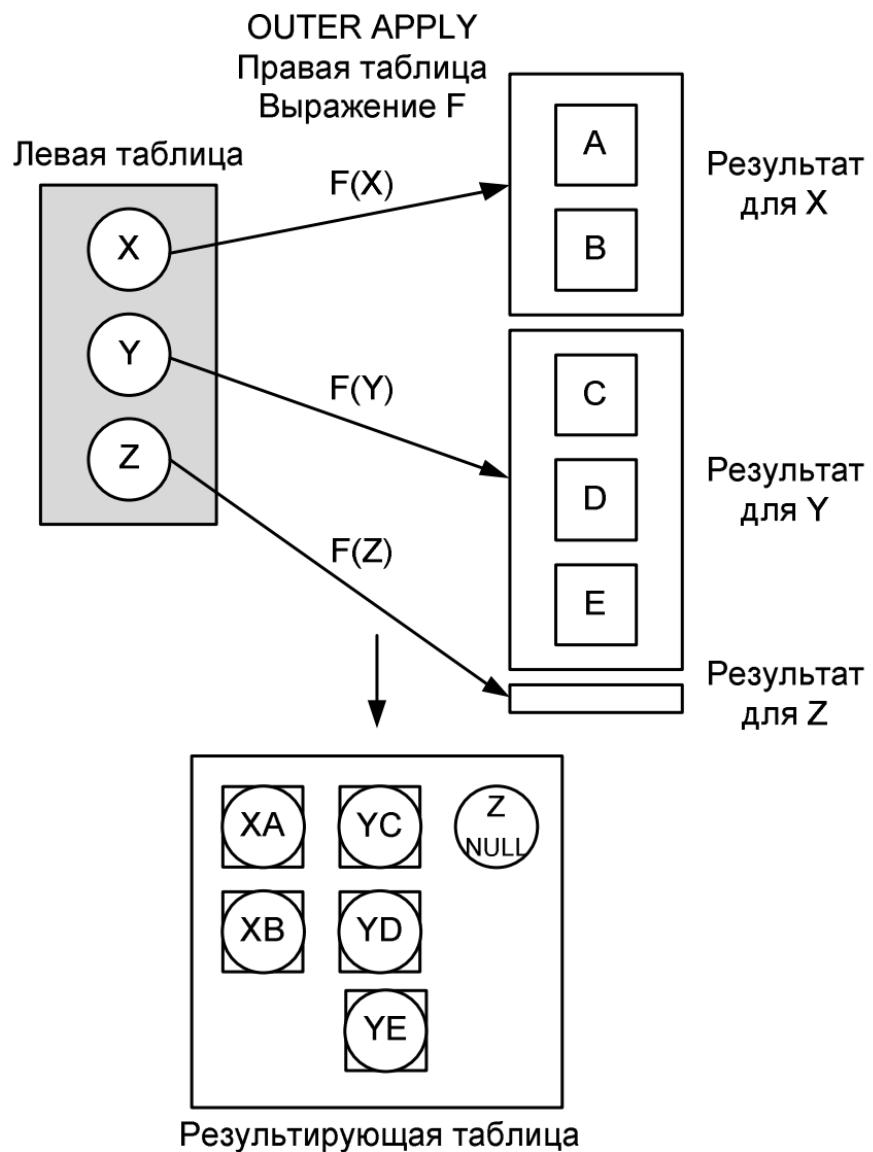
По сравнению с объединением, оператор APPLY интересен тем, что правое табличное выражение может быть связано с левой таблицей; другими словами, внутренний запрос в правом табличном выражении может иметь ссылку на элемент из левой таблицы.

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.



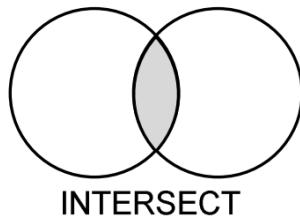
ВАЖНО!

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.

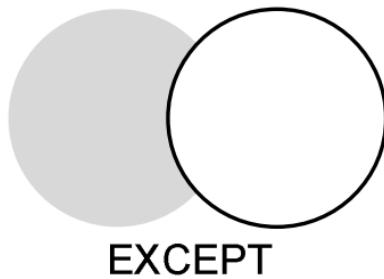
4.5.2 Оператор INTERSECT

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. ($<query 1> \text{ UNION } <query 2>$) INTERSECT ($<query 3> \text{ EXCEPT } <query 4>$).
 - B. $<query 1> \text{ UNION } (<query 2> \text{ INTERSECT } <query 3>)$ 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: B.
 - 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: A.
 - 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).

▼ Вспомогательная

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.

660

Ответы

- 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 XML данные.
- 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

верен или неверен, можно пакти в приложении Ответы 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 D.
 - 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.