



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	3
1.1 Основы языка T-SQL	3
1.2 Понимание логической обработки запросов	6
2 Начало работы с инструкцией SELECT	12
2.1 Использование FROM и SELECT	12
2.2 Работа с типами данных и встроенными функциями	15
2.2.1 Выбор типов данных для ключей	16
2.2.2 Выражение CASE и связанные с ним функции	18
2.2.3 Функция COALESCE	18
3 Фильтрация и сортировка данных	26
3.1 Фильтрация данных с помощью предикатов	26
3.1.1 Предикат LIKE	27
3.1.2 Фильтрация данных даты и времени	28
3.2 Сортировка данных	32
3.3 Фильтрация данных с помощью TOP и OFFSET...FETCH . . .	36
3.3.1 Фильтрация данных с помощью предложения TOP . . .	36
3.3.2 Фильтрация данных с помощью OFFSET...FETCH . . .	37

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.

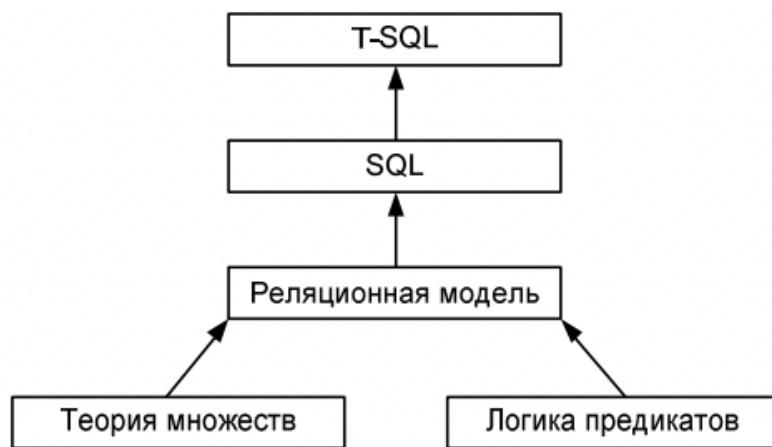


Рис. 1.1: Эволюция языка T-SQL

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.