МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ
ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ
ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ, МЕХАНИКИ И ОПТИКИ

Т.В. Зудилова, Г.Ю. Шмелева

Создание запросов в Microsoft SQL Server 2008

Учебное пособие

Санкт-Петербург
2013

УДК 004.655, 004.657, 004.62

Т.В. Зудилова, Г.Ю. Шмелева

Создание запросов в Microsoft SQL Server 2008 - СПб: НИУ ИТМО, 2013. – 149 с.

В пособии излагаются методические рекомендации к выполнению лабораторных работ по дисциплине “Создание клиент-серверных приложений”.
Предназначено для студентов, обучающихся по всем профилям подготовки бакалавров направления: 210700 Инфокоммуникационные технологии и системы связи.
Рекомендовано к печати Ученым советом факультета Инфокоммуникационных технологий, протокол №3 от 19 марта 2013 г.

В 2009 году Университет стал победителем многоэтапного конкурса, в результате которого определены 12 ведущих университетов России, которым присвоена категория «Национальный исследовательский университет». Министерством образования и науки Российской Федерации была утверждена программа его развития на 2009–2018 годы. В 2011 году Университет получил наименование «Санкт-Петербургский национальный исследовательский университет информационных технологий, механики и оптики».

Оглавление

Введение

Курс предназначен для студентов, обучающихся по всем профилям подготовки бакалавров направления: 210700 Инфокоммуникационные технологии и системы связи.

Задачи дисциплины

После изучения данной дисциплины студенты смогут:

- использовать средства запросов;
- создавать SELECT запросы для доступа к данным;
- объединять данные из нескольких таблиц;
- выполнять группировку и суммирование данных с использованием языке Transact-SQL;
- писать запросы на доступ и модификацию данных с использованием подзапросов;
- модифицировать данные в таблицах;
- описывать создание программных объектов;
- создавать сложные запросы.

Предварительные требования

Для прохождения данной дисциплины студенты должны иметь:

- базовые знания операционной системы Microsoft Windows и ее основных возможностей;
- иметь представление о логическом и физическом проектировании базы данных;
- иметь представление о концепциях целостности данных;
- понимать связи между таблицами и колонками (первичный ключ, внешний ключ, связи один к одному, один ко многим, многие ко многим),

Пособие включает методические указания по выполнению лабораторных работ.

1. Начало работы с базами данных и Transact-SQL в SQL Server 2008

1.1. Обзор SQL Server 2008

Архитектура клиент/сервер

Архитектура клиент / сервер предполагает использование клиента для подключения к серверу для обработки данных. Microsoft ® SQL Server ® 2008 может работать в режиме сервера, может использоваться как инструмент разработки на рабочей станции, где сервер и все его компоненты установлены локально, так режиме клиент/сервера приложений, где базы данных хранятся на сервере, а приложение подключается к серверу для реализации серверного процесса.

Архитектура клиент/сервер построена так, чтобы базы данных находились на центральном компьютере, известном как сервер, и были доступны ряду пользователей. Пользователи получают доступ к серверу через клиента или сервер приложений.

В двухуровневой архитектуре клиент/сервер (Рис.1), пользователи запускают приложения на локальном компьютере клиенте, который соединяется по сети с сервером под управлением SQL Server. Клиентское приложение отрабатывает и бизнес-логику и код для вывода на экран данных для пользователя(толстый клиент).

В многоуровневой архитектуре (Рис.1.), сервер приложений обрабатывает и бизнес-логику и взаимодействует с сервером баз данных SQL.

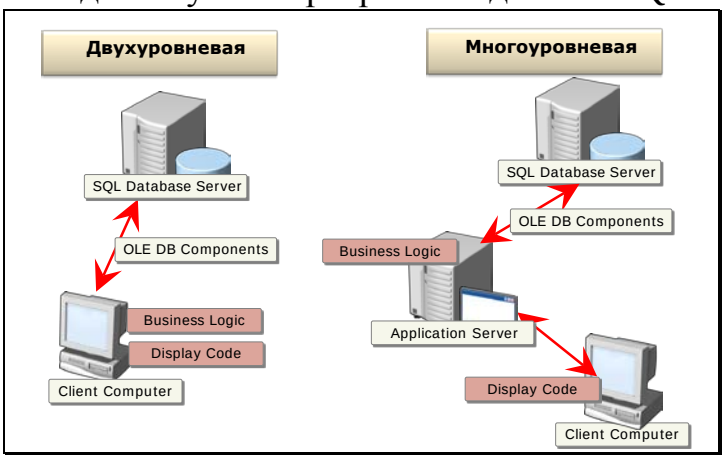


Рис.1. Двухуровневая и многоуровневая архитектура.

Компоненты SQL Server

При установке SQL Server ни один из компонентов не будет установлен, т.к. не выбран. Чтобы компоненты были установлены, их следует выбрать на странице «Выбор компонентов» мастера установки.

Основным компонентом является SQL Server Database Engine, в состав которого входит компонент **Database Engine**, это служба для хранения, обработки и обеспечения безопасности данных, репликации, полнотекстового поиска и средств управления реляционными и XML-данными.

Службы **Analysis Services** содержат средства создания и управления приложениями интерактивной аналитической обработки (OLAP) и приложениями интеллектуального анализа данных.

Службы **Reporting Services** включают в себя серверные и клиентские компоненты для создания, управления и развертывания табличных, матричных и графических отчетов, а также отчетов в свободной форме, службы можно использовать для разработки приложений отчетов.

Службы **Integration Services** представляют собой набор графических средств и программируемых объектов для перемещения, копирования и преобразования данных.

Full-Text Search (полнотекстовый поиск) содержит функциональность, необходимую для выполнения полнотекстовых запросов к простым символьным данным в таблицах SQL Server. Полнотекстовые запросы могут включать слова и фразы, несколько форм слова или фразы. Полнотекстовый поиск позволяет быстро и гибко индексировать текстовые данные, хранящихся в базе данных Microsoft SQL Server для поискового запроса.

Replication (репликация) – представляет собой набор технологий копирования и распространения данных и объектов между базами данных, а также синхронизации баз данных для поддержания согласованности. Используя репликацию, можно распространять данные в различные расположения, а также удаленным или мобильным пользователям по локальным или глобальным сетям посредством коммутируемого соединения, по беспроводным соединениям и через Интернет.

Service Broker обеспечивает SQL Server Database Engine со встроенной поддержкой обмена сообщениями и очередями приложений. Это облегчает разработчикам создание сложных приложений, использующих компоненты Database Engine для связи между разнородными базами данных.

Notification Services представляют собой платформу для разработки приложений, формирующих и отправляющих уведомления, и является движком, запускающим эти приложения.

Средства управления SQL Server

Среда **SQL Server Management Studio** представляет собой интегрированную среду для доступа, настройки, управления, администрирования и разработки компонентов SQL Server. Среда Management Studio позволяет работать с SQL Server разработчикам и администраторам любого уровня подготовки.

Диспетчер конфигурации SQL Server обеспечивает базовые возможности управления конфигурациями для служб, серверных протоколов, клиентских протоколов и псевдонимов клиентов SQL Server.

SQL Server Profiler предоставляет графический пользовательский интерфейс для наблюдения за экземпляром компонента Database Engine или служб Analysis Services.

Помощник по настройке ядра СУБД помогает создавать оптимальные наборы индексов, индексированных представлений и секций.

Среда **Business Intelligence Development Studio** представляет собой интегрированную среду разработки для решений служб Analysis Services, Reporting Services и Integration Services.

Компоненты связи - устанавливает компоненты для связи между клиентами и серверами и сетевые библиотеки для DB-библиотеки, ODBC и OLE DB.

Компоненты SQL Server Databases Engine

Компонент Database Engine представляет собой основную службу для хранения, обработки и защиты данных, обеспечивает управляемый доступ и быструю обработку транзакций для удовлетворения потребностей приложений обработки данных.

Компонент Database Engine можно использовать для создания реляционных баз данных, оперативной обработки транзакций, оперативной аналитической обработки данных.

Протоколы.

Используются для реализации внешнего интерфейса для SQL Server. Для подключения к SQL Server Database Engine должен быть включен сетевой протокол.

Microsoft SQL Server может обслуживать запросы по нескольким протоколам одновременно. Клиенты подключаются к SQL Server с помощью одного протокола.

Relational Engine. Основными функциями Relational Engine являются:

• Парсинг SQL выражений. Анализатор сканирует SQL выражение и разбивает его на логические единицы, такие как ключевые слова, параметры, операторы и идентификаторы. Анализатор также разбивает SQL выражения на несколько меньших логических операций.

• Оптимизация выполнения планов. Как правило, существует много способов, которые сервер может использовать, обрабатывая данные из исходных

таблиц для построения результирующего набора. Оптимизатор запросов определяет, различные серии шагов, оценивает стоимость каждой серии, и выбирает ряд шагов, которые наименее затраты. Затем определяет конкретные шаги в дерева запроса для получения оптимизированного плана выполнения.

• Выполнение ряда логических операций, определенных в плане выполнения. После того, как оптимизатором запросов определены логические операции, необходимые для выполнения выражения, выполняется последовательность шагов, указанная в оптимизированном плане выполнения.

• Обработка данных и операторов создания объектов и подключений. Ряд операторов, помимо SELECT, INSERT, UPDATE или DELETE требуют особой обработки. Примеры представляют собой набор операторов по установлению параметров соединения SET, а так же операторы CREATE создания объектов в базе данных.

• Форматирование результатов. Клиенту возвращаются результаты реляционных данных либо в обычном табличном результирующем наборе или в виде XML-документа. Результаты могут быть заключены в один или более пакетов потоков табличных данных (TDS) и пакеты возвращаются приложению.

Storage Engine.

. Основными функциями механизма хранения являются:

• обеспечение удобства использования управления системами хранения компонентов,

• управление данными буферов, и всеми I / O к физическим файлам,

• контроль параллелизма, управления транзакциями, блокировкой и логированием,

• управление файлами и физическими страницами, используемыми для хранения данных,

• восстановление системных ошибок

SQLOS

SQLOS на уровне пользователя легко конфигурируется операционной системой с мощным API, что позволяет организовать автоматическое ограничение и повышение параллелизма. SQLOS скрывает сложность базового оборудования, высокого уровня программирования, и в то же время она обеспечивает мощный и полный набор функций для программистов, желающих воспользоваться преимуществами аппаратного уровня.

SQLOS обеспечивает операционную систему такими услугами как: отсутствием упреждающего планирования, управлением памятью, обнаружением тупиковых ситуаций, обработкой исключений, хостингом для внешних компонентов, таких, как среды CLR, CLR и другие услуги.

SQLOS моделирует внутреннее состояние системы, чтобы упростить процесс разработки для конечных платформ.

Подсистема планирования SOS состоит из планирования узлов, планировщиков, задач, рабочих и системных потоков. Планирование узла обеспечивает уровень абстракции над группой процессоров.

1.2. Обзор баз данных SQL Server

Обзор реляционных БД

Реляционные базы данных хранят данные в ряде взаимосвязанных таблиц. Как правило, таблицы в реляционной базе данных имеют отношение один-ко-многим. Например, реляционная база данных может иметь одну таблицу в которой хранятся заказы (Рис.2.), а другую, которая хранит позиции каждого заказа. Каждый заказ в таблице будет иметь одну или несколько позиций в таблице сведений о заказах. Другим примером может служить таблица клиентов и заказов таблице. Каждый клиент может разместить несколько заказов.

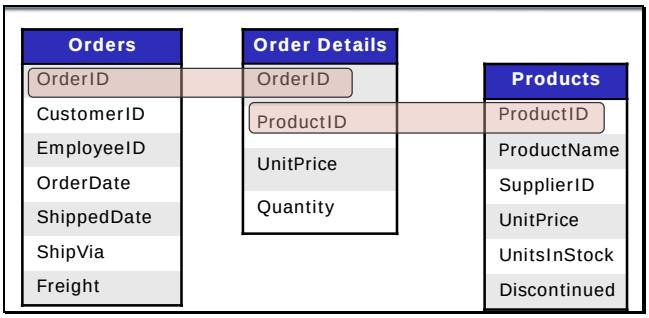


Рис.2. Реляционные связи между таблицами.

SQL Server имеет две основные части: инструмент реализации реляционной БД и инструмент хранения. Оба инструмента работают независимо, взаимодействуя друг с другом посредством собственного доступа к данным компонентов, таких как связывание и внедрение объектов, базы данных (OLE DB).

Объектами базы данных в реляционной базе данных являются таблицы, индексы, представления, процедуры и другие объекты, рассмотренные ниже.

Нормализация

Избыточные данные являются лишними на диске и создают проблемы с обслуживанием. Если данные, которые существуют в более чем одном месте, должны быть изменены, то они должны быть изменены точно так же во всех остальных местах. Изменение адреса клиента гораздо легче реализовать, если эти данные хранятся только в таблице клиентов и больше нигде в базе данных.

Для устранения проблем дублирования данных, при разработке схемы базы данных используют нормализацию, которую определяют как процесс удаления избыточных данных.

Нормализация часто повышает производительность. Но с нормализацией, растет количество и сложность соединений, необходимых для получения данных. В качестве правила Microsoft предлагает выполнять процесс нормализации, если приложение вызывает много запросов.

Есть несколько правил нормализации баз данных. Каждое правило называется «нормальной формой». Если реализовано первое правило, то база данных называется в «первой нормальной форме». Если реализованы первые три правила, что база данных считается в «третьей нормальной форме». Несмотря на то, что возможны несколько уровней нормализации, третья нормальная форма считается уровнем, достаточным необходимым для большинства приложений.

Нормализация включает в себя создание таблиц и установление отношений между этими таблицами в соответствии с правилами, предназначенными как для защиты данных, так и для того, чтобы сделать базу данных более гибкой, устраняя избыточность и несогласованные зависимости.

Логическое проектирование баз данных с использованием нормализации дает лучшую производительность. Большее число небольших (узких) таблиц характерно для нормализованной базы данных. Большее количество больших (широких) таблиц характерно для денормализованных баз данных.

Высоко нормализованные базы данных, связанные с комплексными реляционными соединениями, могут снизить производительность. Оптимизатор SQL Server является эффективным при выборе соединений, если доступны индексы.

Преимущества нормализации:

- Ускорение сортировки и создания индекса по небольшим таблицам.
- Создание кластерных индексов для несколько таблиц.
- Меньшее количество индексов в таблице, улучшает производительность при UPDATE.
- Меньше NULL значений и меньше избыточных данных, увеличивает компактность базы.

Недостатки нормализации:

- Больше таблиц присоединяется при выборке данных (разделение данных в несколько таблиц, увеличивает необходимость объединения таблиц).
- Получение данных и запросов может работать медленнее из-за объединения.
- Таблицы содержат коды, а не реальные данные (в случае повторения данные хранятся в виде кодов, а не реальных данных - всегда есть необходимость перехода на таблицу поиска для значения).
- Данные трудно запросить без модели (модель данных оптимизирована для приложений, а не для специальных запросов).

Процесс нормализации

Рассмотрим определение первых трех форм нормализации. Эти нормы обычно реализуются для организации данных в базе данных.

Для приведения базы данных к той или иной нормальной форме требуется выполнить ряд правил.

Первая нормальная форма

- Устраните повторяющиеся группы в отдельных таблицах.
- Создайте отдельную таблицу для каждого набора связанных данных.
- Определите в каждой таблице набор данных, связанных с первичным ключом.

Например, в первой нормальной форме находятся две таблицы: таблица заказов и таблица деталей заказа (Рис.3.).

Вторая нормальная форма

• Создайте отдельные таблицы для наборов значений, относящихся к нескольким записям.

• Свяжите эти таблицы с внешним ключом.

Третья нормальная форма

• Устраните поля, которые не зависят от ключа.

• Транзитивные зависимости должны быть устранены, так что все записи должны полагаться только на первичный ключ.

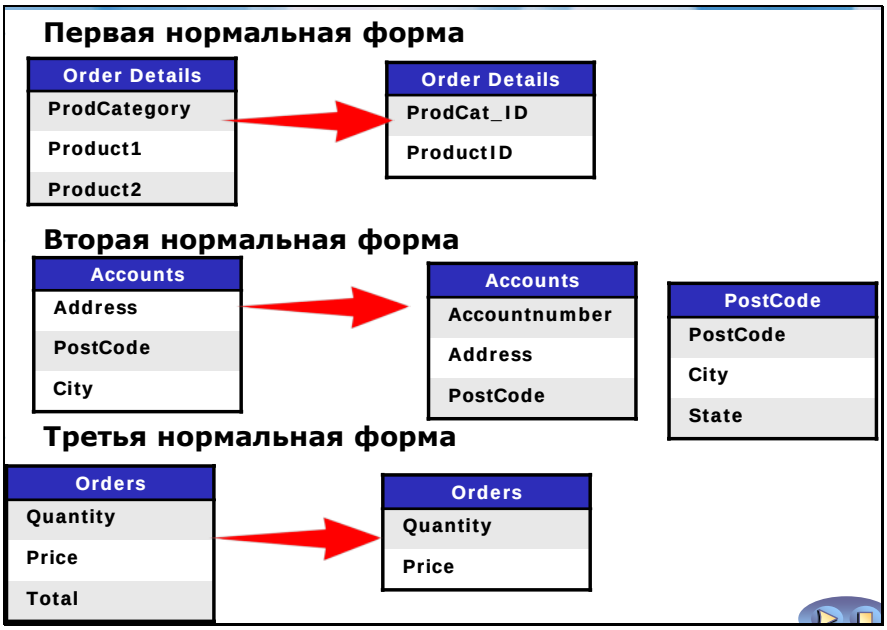


Рис.3. Таблицы в нормальных формах.

Объекты базы данных SQL Server

Таблицы - являются основной формой для сбора информации, содержат все данные в базах данных SQL Server. Каждая таблица представляет собой тип объекта, который имеет смысл для пользователей.

Индекс – структура на диске, связанная с таблицей или представлением, которая ускоряет поиск строк таблицы или представления. Индекс содержит ключи, созданные для одного или нескольких столбцов таблицы или представления. Эти ключи хранятся в виде B-древовидной структуры, что позволяет SQL Server, быстро и эффективно находить строку или строки, связанные с ключевыми индексами. Существуют кластерные и не кластерные индексы.

Представление - виртуальная таблица или хранимый запрос. Данные доступные через представление хранятся в базе данных не как отдельный объект, а как выражение SELECT, в результате SELECT формирует виртуальную таблицу, возвращая представление. Пользователь может использовать эту виртуальную таблицу, ссылаясь на представление в Transact-SQL (T-SQL) так же, как на таблицы ссылок.

Хранимые процедуры в Microsoft SQL Server, аналогичны процедурам в языках программирования относительно их действий:

- принимают входные параметры для вызова процедуры или пакета и возвращать несколько значений в виде выходных параметров,
- содержат выражения программирования, которые выполняют операции в базе данных, включая вызов других процедур,
- возвращают значение состояния вызывающей процедуре или пакету, чтобы указать, выполнение или не выполнение (и причины отказа).

Для запуска хранимой процедуры используют выражение EXECUTE языка Transact-SQL. Хранимые процедуры отличаются от функций тем, что могут не возвращать значения в место их вызова и поэтому и они не могут быть использованы непосредственно в выражениях.

Триггеры представляют собой объекты базы данных, связанные с таблицей. Во многом они похожи на хранимые процедуры и часто упоминаются как "особый вид хранимых процедур". Основное различие между триггером и хранимой процедурой в том, что триггер связан с таблицей и работает только при работе выражения INSERT, UPDATE или DELETE.

Основная работа по **ограничениям** является соблюдением правил в базе данных, предназначенных для обеспечения целостности данных. Например, у нас есть ограничения внешнего ключа, чтобы убедиться, что все заказы ссылаются на существующую продукцию. Поддержание целостности имеет первостепенное значение для базы данных, т.к. мы не можем доверять пользователям и приложения и быть уверенными, эти правила будут соблюдены. Если целостность нарушается, то возможны ситуации, когда у клиентов будет двойной счет, расчетов с поставщиком не хватает, что приводит к потере уверенности в работе приложения.

Правило определяет допустимые значения, которые могут быть вставлены этот столбец.

Обзор типов данных

В SQL Server, столбцы, локальные переменные, выражения, и параметры имеют соответствующий тип данных.

Тип данных атрибута, который определяет тип данных столбца, может быть следующих типов: целочисленные данные, символьные данные денежный тип, тип даты и времени, двоичные строки, и так далее.

SQL Server предоставляет набор системных типов данных, которые определяют все типы данных, которые могут быть использованы с SQL Server.

Можно также определить собственные типы данных в Transact-SQL или Microsoft. NET Framework, на основе системного типа данных. Пользовательские типы получают их характеристики от методов и операторов класса, который создается с помощью одного из поддерживаемых .NET Framework языков программирования. SQL Server предоставляет совместимые типы данных (ISO).

Когда два выражения, которые имеют различные типы данных, сортировку, точность, масштаб и длину вложенных операторов, характеристики результирующего типа определяются следующим образом:

- Тип данных результата определяется применением правил преобразования в типы данных входного выражения.
- Сортировка результата определяется правилами очередности параметров сортировки, если тип данных результата CHAR, VARCHAR, текст, NCHAR, NVARCHAR или NTEXT.
- Точность, масштаб и длина результата зависят от точности, масштаба и длины входных выражений.

Типы данных в SQL Server, организованных в следующие категории:

- точные числовые: bigint, numeric, bit, smallint, decimal, smallmoney, int, tinyint, money;
- символьные строки в Юникоде: nchar, nvarchar, ntext;
- приблизительные числовые: float, real;
- двоичные: binary, varbinary, image;
- дата и время: date, datetimeoffset, datetime2, smalldatetime, datetime, time;
- символьные строки: char, varchar, text;
- другие типы данных: cursor, timestamp, hierarchyid, uniqueidentifier, sql_variant, xml, table.

В SQL Server, в зависимости от их параметров хранения, некоторые типы данных отнесены к следующим группам:

•большие значения типов данных: varchar(max), nvarchar(max), varbinary(max)

• типы данных больших объектов: text, ntext, image, varchar(max), nvarchar(max), varbinary(max), and xml

1.3. T-SQL

Синтаксис T-SQL

SELECT извлекает данные из SQL Server, и возвращает их пользователю в одном или нескольких результирующих наборах в виде таблицы. Как и таблицы SQL, результирующий набор состоит из столбцов и строк.

Полный синтаксис оператора выбора является сложным, но большинство SELECT выражений описывают четыре основных свойства результирующего набора:

1) количество и атрибуты столбцов в наборе результатов. Следующие атрибуты должны быть определены для каждого столбца результирующего набора:

• тип данных столбца,

• размер столбца, для числовых столбцов, точность и размер,

• источник данных возвращаемых значений в столбце.

2) таблицы, из которых результирующий набор данных извлекается, и любые логические связи между таблицами.

3) условия, которым должны соответствовать строки в исходных таблицах, чтобы попасть в выборку на SELECT. Строки, которые не отвечают условиям - игнорируются.

4) последовательность, в которой строки результирующего набора упорядочены.

Пример:

```
SELECT ProductID, Name, ListPrice
FROM Production.Product
WHERE ListPrice > $40
ORDER BY ListPrice ASC
```

Типы операторов T-SQL

Операторы T-SQL делятся на несколько типов:

Тип	Операторы
Арифметические операторы	+, -, *, /, % Vacation + SickLeave AS 'Total PTO'

Операторы присваивания	= SET @MyCounter = 1
Операторы сравнения	=, <, >, <>, !, >=, <= IF (@MyProduct <> 0) …
Логические операторы	AND, OR, NOT WHERE Department = ‘Sales' AND (Shift = 'Evening' OR Shift = 'Night')
Оператор слияния строк	+ SELECT LastName + ', ' + FirstName AS Moniker

Арифметические операторы используются для математических функций.

Оператор присваивания используется для присваивания значения.

Операторы сравнения могут быть использованы в выражениях с любыми типами данных, кроме text, ntext, image.

Логические операторы используются, для указания нескольких условий поиска объединены в поисковом запросе. Логические операторы позволяют создавать более сложные выражения поиска из простых, и тем самым сузить область поиска. Скалярные функции могут быть использованы там, где логическое выражение справедливо.

Объединение строк осуществляется с помощью +. Оператор в строковом выражении объединяет две или более символьных или двоичных строки, столбца или комбинацию строк и столбцов в одно выражение.

Функции T-SQL

Рассмотрим несколько типов функций в T-SQL:

Функции	**Описание**
Наборов строк	Возвращают объект, который можно использовать вместо ссылки на таблицу

Пример: CONTAINSTABLE, OPENDATASOURCE, OPENQUERY	
Статистические	Выполняют вычисление на наборе значений и возвращают одиночное значение
Пример: AVG, CHECKSUM_AGG, SUM, COUN	
Ранжирующие	Возвращают ранжирующее значение для каждой строки в секции
Пример: RANK, DENSE_RANK	
Скалярные	Обрабатывают и возвращают одиночное значение
Пример: CREATE FUNCTION dbo.ufn_CubicVolume	

Функции набора строк - возвращают объект, который можно использовать как таблицу ссылок в выражении SQL.

```
SELECT select_list FROM table AS FT_TBL INNER JOIN
CONTAINSTABLE(table, column, contains_search_condition) AS KEY_TBL ON
FT_TBL.unique_key_column = KEY_TBL.[KEY]
```

Статистические функции – обрабатывают наборы значений, но возвращают одно значение.

```
SELECT AVG(VacationHours)as 'Average vacation hours', SUM (SickLeaveHours)
as 'Total sick leave hours' FROM HumanResources.Employee WHERE Title LIKE
'Vice President%';
```

Функции ранжирования - возвращают рейтинг значений для каждой строки в секции

```
GO
SELECT c.FirstName, c.LastName
   ,ROW_NUMBER() OVER (ORDER BY a.PostalCode) AS 'Row Number‘
   ,RANK() OVER (ORDER BY a.PostalCode) AS 'Rank‘
   ,DENSE_RANK() OVER (ORDER BY a.PostalCode) AS 'Dense Rank‘
   ,NTILE(4) OVER (ORDER BY a.PostalCode) AS 'Quartile‘
   ,s.SalesYTD, a.PostalCode
FROM Sales.SalesPerson s
    INNER JOIN Person.Contact c
       ON s.SalesPersonID = c.ContactID
   INNER JOIN Person.Address a
       ON a.AddressID = c.ContactID
WHERE TerritoryID IS NOT NULL
    AND SalesYTD <> 0;
```

Скалярные функции – обрабатывают одно значение, и возвращают одно значение.

```
SELECT ProductModelID, Name, dbo.ufnGetInventoryStock(ProductID)AS
CurrentSupply
FROM Production.Product
WHERE ProductModelID BETWEEN 75 and 80;
```

Переменные T-SQL

Переменные предназначены для хранения значений.

Переменная имеет тип данных. Используемые переменные следует объявить, а затем заполнить значением.

В Transact-SQL локальные переменные являются объектами, которые могут содержать одно значение определенного типа данных.

Переменные в пакетах и сценариях обычно используются, если:

• требуется контролировать количество выполнений цикла,

• проверки контрольных значений,

• для сохранения значения данных, которые будут возвращены в точку вызова хранимой процедуры или возвращаемого функцией значения.

Оператор Declare объявляет переменную в Transact-SQL:

• присваивает имя переменной, имя должно иметь в качестве первого символа - @

• объявляет системный или пользовательский тип данных и длину. Для числовых переменных назначает точность и размер. Для переменных типа XML могут быть назначены дополнительные схемы

• устанавливает значение NULL.

Когда переменная объявлена, ее значение равно NULL. Чтобы присвоить значение переменной, используется оператор SET. Это наиболее предпочтительный метод присваивания значения переменной. Переменная может так же иметь значение, которое присвоится ей в результате выбора SELECT.

Чтобы присвоить переменной значение с помощью SET, следует указать имя переменной и присваиваемое переменной значение. Переменная может иметь значение, назначенное на которые ссылаются в списке выбора. Если SELECT возвращает одно значение, переменная она должна быть скалярной. Если оператор SELECT, возвращает более одной строки, переменная должна быть не скалярной, переменной присваивается возвращенное значение результирующего набора.

Переменная может быть установлена динамически, с помощью кода, аналогичного следующему:

```
USE Northwind
DECLARE @sum AS decimal(10, 4)

UPDATE [Order Details]
  SET UnitPrice = UnitPrice * 1.1
  WHERE OrderID = 10248

SET @sum = (SELECT SUM(Quantity * UnitPrice)
  FROM [Order Details]
  WHERE OrderID = 10248)

SELECT @sum
```

Выражения T-SQL

Выражения T-SQL это сочетание символов и операторов, используемое для вычисления одиночного значения данных. Отдельные константы, переменные, столбцы и скалярные функции являются примерами простых выражений. Для соединения двух и более простых выражений в одно сложное можно использовать операторы.

Если простое выражение состоит из отдельной константы, переменной, скалярной функции или имени столбца, то тип данных, параметры сортировки, точность, масштаб и значение выражения совпадают с типом данных, параметрами сортировки, точностью, масштабом и значением ссылаемого объекта.

При объединении двух выражений с помощью операторов сравнения или логических операторов результат будет иметь логический тип и может принимать одно из следующих значений: TRUE, FALSE или UNKNOWN.

При объединении двух выражений с помощью арифметических, побитовых или строковых операторов тип данных результата определяется используемым оператором.

Сложные выражения, составленные из нескольких символов и операторов, вычисляются в одиночные результаты. Тип данных, параметры сортировки, точность и значение результирующего выражения определяются объединением составляющих выражений (по два за один раз) до тех пор, пока не будет получен конечный результат. Последовательность, в которой эти выражения объединяются, зависит от приоритета операторов в выражении.

Два выражения можно объединить каким-либо оператором, если оба они содержат данные типов, поддерживаемых оператором, и выполняется хотя бы одно из следующих условий.

Выражения содержат данные одинаковых типов.

Тип данных с более низким приоритетом может быть неявно преобразован в тип данных с более высоким приоритетом.

Если выражения не удовлетворяют этим условиям, функция CAST или CONVERT явным образом преобразует тип данных с более низким приоритетом или в тип данных с более высоким приоритетом, или в промежуточный тип данных, который затем может быть неявно преобразован в тип данных с более высоким приоритетом.

Если неявное или явное преобразование не поддерживается, эти два выражения объединить невозможно.

Параметры сортировки любого выражения, результатом которого является символьная строка, определяются правилами очередности параметров сортировки.

Управляющие элементы

Язык Transact-SQL содержит специальные ключевые слова, известные как язык управления выполнением, которые управляют порядком выполнения инструкций на языке Transact-SQL, блоками инструкций, определяемыми пользователем функциями и хранимыми процедурами.

Без языка управления выполнением отдельные инструкции языка Transact-SQL выполнялись бы последовательно — так, как они написаны. Язык управления выполнением позволяет связывать инструкции друг с другом, а также создавать независимо выполняющиеся конструкции, как в языках программирования.

Ключевые слова управления выполнением полезны в тех случаях, когда с помощью языка Transact-SQL необходимо выполнить какое-либо действие.

Например, пара инструкций **BEGIN...END** предназначена для объединения нескольких инструкций языка Transact-SQL в логический блок.

Пара инструкций **IF...ELSE** окажется полезной, если инструкцию или блок инструкций необходимо выполнять только при соблюдении каких-либо условий, а другую инструкцию или группу инструкций — в противном случае (условие ELSE).

Инструкции управления выполнением не могут быть распределены по разным пакетам, определяемым пользователем функциям или хранимым процедурам.

Инструкция **WAITFOR** приостанавливает выполнение пакета, хранимой процедуры или транзакции до тех пор, пока:

не пройдет заданный интервал времени;

не наступит заданное время суток;

инструкция **RECEIVE** не изменит или не возвратит, по крайней мере, одну строку в очередь компонента Service Broker.

Использование WAITFOR:

• WAITFOR TIME '22: 20 ';

• WAITFOR DELAY '02: 00 ';

1.4. Работа со скриптами T-SQL

Пакеты

Пакет является группой из одной или нескольких инструкций языка Transact-SQL, отправляемых единовременно из приложения в SQL Server для выполнения. Сервер SQL Server компилирует инструкции пакета в единый исполняемый модуль, называемый планом выполнения. Инструкции в плане выполнения затем последовательно выполняются.

Каждая инструкция Transact-SQL должна заканчиваться точкой с запятой. Сейчас это не обязательно, но против инструкций без точки с запятой есть серьезные возражения, и в последующих версиях Microsoft SQL Server такой возможности может не быть.

Ошибка компиляции, например, синтаксическая ошибка, прекращает компиляцию плана выполнения. Поэтому никакие инструкции в пакете не выполняются.

Ошибка выполнения, например арифметическое переполнение или нарушение ограничения, приводит к одному из двух результатов:

Большинство ошибок времени выполнения останавливают текущую инструкцию и инструкции, следующие за ней в пакете.

Некоторые ошибки времени выполнения, например, нарушения ограничений, останавливает только текущую инструкцию. Все остальные инструкции в пакете будут выполнены.

Это не оказывает влияния на инструкции, которые выполняются до той инструкции, в которой встретилась ошибка выполнения. Единственное исключение — если пакет находится в транзакции, и из-за ошибки будет выполнен откат транзакции. В этом случае будет выполнен откат любых незафиксированных изменений данных, сделанных перед ошибкой выполнения.

Например, предположим, что в пакете имеется 10 инструкций. Если пятая инструкция содержит синтаксическую ошибку, никакие инструкции в пакете не выполнятся. Если пакет скомпилирован, и затем вторая инструкция завершается неудачно при выполнении, это не повлияет на результаты первой инструкции, потому что она уже выполнена.

GO - директива, которая информирует программы SQL Server об окончании пакета инструкций Transact-SQL.

```
EXECUTE usp_GetList '%Bikes%', 700,
   @compareprice OUT,
   @cost OUTPUT
IF @cost <= @compareprice
BEGIN
   PRINT 'These products can be purchased for less than
   $'+RTRIM(CAST(@compareprice AS varchar(20)))+'.'
END
ELSE
   PRINT 'The prices for all products in this category exceed
   $'+ RTRIM(CAST(@compareprice AS varchar(20)))+'.‘
GO
```

Обработка ошибок

При написании кода следует планировать обработку ошибок и исключений.

Группа инструкций на языке Transact-SQL может быть заключена в блок TRY. Если ошибка возникает в блоке TRY, управление передается следующей группе инструкций, заключенных в блок CATCH.

За блоком TRY сразу же должен следовать блок CATCH. Размещение каких-либо инструкций между инструкциями END TRY и BEGIN CATCH вызовет синтаксическую ошибку.

Конструкция TRY…CATCH не может охватывать несколько пакетов. Конструкция TRY…CATCH не может охватывать множество блоков инструкций на языке Transact-SQL.

Конструкция TRY...CATCH может быть вложенной. Либо блок TRY, либо блок CATCH могут содержать вложенные конструкции TRY…CATCH. На-

пример: блок CATCH может содержать внутри себя внедренную TRY…CATCH для управления ошибками, возникающими в коде CATCH.

Конструкция TRY…CATCH не может использоваться в пользовательских функциях.

```
BEGIN TRY
   -- Generate divide-by-zero error.
   SELECT 1/0;
END TRY
BEGIN CATCH
   -- Execute error retrieval routine.
   EXECUTE usp_GetErrorInfo;
END CATCH;
```

Инструкция RAISERROR создает сообщение об ошибке и запускает обработку ошибок для сеанса, может либо ссылаться на определенное пользователем сообщение, находящееся в представлении каталога sys.messages, либо динамически создавать сообщение. Это сообщение возвращается как сообщение об ошибке сервера вызывающему приложению или соответствующему блоку CATCH конструкции TRY…CATCH.

```
RAISERROR (N'This is message %s %d.', -- Message text.
          10, -- Severity,
          1, -- State,
          N'number', -- First argument.
          5); -- Second argument.
-- The message text returned is: This is message number 5.
GO
```

Комментарии

Комментарий отображает текст, введенный пользователем. Текст, помещенный между /* и */, не вычисляется сервером.

Комментарии могут вставляться в отдельную строку или в пределах инструкции Transact-SQL. Многострочные комментарии необходимо отмечать сочетаниями символов /* и */. Для многострочных комментариев часто используется следующий стиль: первую строку начинают с сочетания символов /*, последующие строки — с сочетания символов **, а заканчивают комментарий сочетанием символов */.

Ограничения на максимальную длину комментариев не существует.

Поддерживаются вложенные комментарии. Если сочетание символов /* используется в пределах существующего комментария, он рассматривается как

начало вложенного комментария и, следовательно, требует метки */, закрывающей этот комментарий. Если метки, закрывающей комментарий, нет, выдается ошибка.

1.5. Использование T-SQL запросов

Инструменты для создания запросов в базах данных

Рассмотрим обзор приложений, которые используются для создания и выполнения запросов SQL Server 2008.

Среда SQL Server Management Studio поддерживает два способа для доступа и изменения данных.

1. Из меню **Файл** и с помощью кнопок **Создать запрос** и **Запрос к ядру СУБД** на панели инструментов можно открыть окно запроса компонента Database Engine. В окне запроса компонента Database Engine можно интерактивно создавать инструкции Transact-SQL и XQuery, чтобы направлять запросы к базам данных и изменять данные. Можно сохранить эти инструкции в виде файлов сценариев и впоследствии запускать их с помощью программы **sqlcmd**. Редактор запросов компонента Database Engine поддерживает динамическую справку F1, автоматическое дополнение, структурирование кода, отладчик Transact-SQL, технологию IntelliSense и другие средства производительности.
2. В обозревателе объектов можно щелкнуть правой кнопкой мыши таблицу или представление и выбрать элементы меню, позволяющие выделять и изменять строки.

Программа **sqlcmd** — это программа командной строки Microsoft Win32 для нерегламентированного, интерактивного запуска инструкций Transact-SQL и XQuery, запуска файлов сценариев Transact-SQL и XQuery. Чтобы пользоваться программой **sqlcmd**, необходимо понимать язык программирования Transact-SQL и XQuery. В программе **sqlcmd** используется API-интерфейс поставщика OLE DB собственного клиента SQL Server. Она заменяет программу командной строки **osql**, в которой используется ODBC API-интерфейсе.

Программа **bcp** используется для вставки большого количества строк в таблицы SQL Server. Для использования этой программы не требуется знания языка Transact-SQL, но необходимо понимание структуры таблиц, в которые копируются новые строки, а также допустимых типов данных для строк таблицы. Программа **sqlps** — это служебная программа командной строки Microsoft C#, предназначенная для:

- нерегламентированного, интерактивного запуска команд PowerShell;
- запуска файлов сценариев PowerShell.

Программа **sqlps** загружает и регистрирует поставщик SQL Server PowerShell. С ее помощью можно перемещаться по моделям управляющих объектов SQL Server с использованием путей, аналогичных путям файловой систе-

мы. С помощью командлета **Run-Sqlcmd** можно запустить файлы сценариев, которые содержат инструкции Transact-SQL и XQuery, поддерживаемые программой **sqlcmd**.

Среду SQL Server Management Studio и программу **sqlps** можно использовать для соединения и администрирования одновременно нескольких экземпляров SQL Server. Программы **sqlcmd** и **bcp** позволяют подключаться только к одному экземпляру SQL Server в данный момент времени.

SQL Server Management Studio

Среда SQL Server Management Studio обеспечивает следующие возможности:

поддерживает большинство административных задач для SQL Server;

- является единой интегрированной средой для управления SQL Server Database Engine и разработки;
- содержит диалоговые окна для управления объектами в компоненте SQL Server Database Engine, службах Analysis Services, Reporting Services, Notification Services и выпуске SQL Server Compact 3.5 с пакетом обновления 1 (SP1), позволяющие выполнять действия немедленно, направлять их в редактор кода или включать эти действия в сценарий для последующего выполнения;
- содержит немодальные диалоговые окна с настройкой размеров, позволяющие при открытом диалоговом окне получать доступ к нескольким средствам;
- содержит общее диалоговое окно планирования, позволяющее выполнять действия управляющих диалоговых окон в заданное время;
- позволяет выполнять экспорт и импорт регистрации сервера среды SQL Server Management Studio из одной среды Management Studio в другую;
- позволяет выполнять сохранение и печать XML-файлов плана выполнения и взаимоблокировок, созданных приложением SQL Server Profiler, просмотр их в любое время и отправка для анализа администратору;
- содержит окна сообщений об ошибках и информационных сообщений, предоставляющие гораздо больше сведений и позволяющие отправлять в Майкрософт комментарии о сообщениях, копировать сообщения в буфер обмена и отправлять их по электронной почте в службу поддержки;
- содержит встроенный веб-обозреватель для быстрого обращения к библиотеке MSDN или получения интерактивной справки;
- содержит встроенную справку от сообществ в Интернете;
- содержит учебник по среде SQL Server Management Studio, облегчающий освоение многих новых возможностей и помогающий сразу правильно и продуктивно их использовать.

- содержит новый монитор активности с фильтрацией и автоматическим обновлением;
- содержит встроенные интерфейсы компонента Database Mail.

Решения SQL Server

Решения и проекты сценариев представляют собой контейнеры, которые разработчики используют для организации связанных файлов в среде SQL Server Management Studio. Управление решениями и проектами сценариев осуществляется с помощью обозревателя решений.

Management Studio можно использовать в качестве платформы для разработки сценариев для служб SQL Server, Analysis Services и SQL Server Compact 3.5 с пакетом обновления 1 (SP1). Используйте Management Studio для разработки сценариев для реляционных и многомерных баз данных и для всех типов запросов. Возможности разработки в Management Studio дополнены широким набором мощных редакторов кода. Приложение Management Studio позволяет:

- создание запросов и сценариев для поддержания производственных процессов;
- добавление в проект сведений о соединении и других связанных с ним файлов;
- сохранение в проекте запросов и сценариев вместе с их соединениями;
- организацию проектов сценариев в одном контейнере, именуемом решением;
- сохранение решения в базе данных Microsoft Visual SourceSafe (VSS) или продуктах других поставщиков системы управления версиями для отслеживания изменений при разработке решения и управления его жизненным циклом.

В среде Microsoft SQL Server Management Studio реализовано два вида контейнеров для управления такими проектами баз данных, как сценарии, запросы, соединения с данными и файлы: решения и проекты. Объекты, содержащиеся в этих контейнерах, называются элементами.

Важно! В будущей версии Microsoft SQL Server эта возможность будет удалена. Избегайте использования этой возможности в новых разработках и запланируйте изменение существующих приложений, в которых она применяется.

Проект — это набор файлов и относящихся к ним метаданных, например сведений о соединении. Состав файлов в проекте зависит от того, для какого из компонентов Microsoft SQL Server этот проект предназначен. Например, проект SQL Server может содержать инструкции DDL, определяющие объекты в базе данных.

Решение содержит один или несколько проектов, а также файлы и метаданные, которые позволяют определить решение как единое целое.

Решения и проекты содержат элементы, которые представляют собой сценарии, запросы, сведения о соединениях и файлы, необходимые для создания решения базы данных. Эти контейнеры позволяют:

- реализовать систему управление версиями для запросов и сценариев;
- управлять параметрами как решения в целом, так и конкретных проектов;
- использовать обозреватель решений, облегчающий работу с файлами и позволяющий сосредоточиться на работе с элементами, составляющими решение базы данных;
- добавлять элементы одновременно к нескольким проектам решения либо ко всему решению, не указывая элемент в каждом из проектов;
- работать с различными файлами, формат которых не зависит от решений и проектов.

Элементы, содержащиеся в проектах, зависят от типа проекта и от того, используется ли среда SQL Server Management Studio. Этот раздел содержит следующие темы.

Среда SQL Server Management Studio не поддерживает решения и проекты Microsoft Visual Studio и Microsoft Business Intelligence Development Studio.

2. Запросы и фильтрация данных.

2.1. Использование оператора SELECT

SELECT используется для получения одной или нескольких строк данных из одной или нескольких таблиц. SELECT только извлекает данные, нет обновления, удаления или вставки выполнены. Единственное исключение, когда используется INTO, результат получают в новой таблице.

Полный синтаксис инструкции SELECT является сложным, однако основные предложения можно вкратце описать следующим образом:

```
SELECT select_list
[ INTO new_table_name ]
FROM table_list
[ WHERE search_conditions ]
[ GROUP BY group_by_list ]
[ HAVING search_conditions ]
[ ORDER BY order_list [ ASC | DESC ] ]
```

select_list

Описывает столбцы результирующего набора. Это список выражений с разделителями-запятыми. Каждое выражение определяет как формат (тип данных и размер), так и источник данных для столбца результирующего набора. Каждое выражение списка выбора обычно ссылается на столбец в исходной

таблице или представлении, предоставляющем данные, но может быть любым другим выражением, например константой или функцией Transact-SQL. Использование выражения ***** в списке выбора указывает, что должны быть возвращены все столбцы исходной таблицы.

INTO new_table_name

Указывает, что результирующий набор используется для создания новой таблицы. Параметр new_table_name указывает имя новой таблицы.

FROM table_list

Содержит список таблиц, из которых будут получены данные результирующего набора. Этими источниками могут быть:

- Базовые таблицы на локальном сервере, где работает SQL Server.
- Представления в локальном экземпляре SQL Server. Внутри SQL Server разрешаются ссылки на представления относительно базовых таблиц, на которых построено представление.
- Связанные таблицы. Это таблицы в источниках данных OLE DB, к которым можно обратиться с помощью SQL Server. Это упоминается как распределенный запрос. К источникам данных OLE DB можно обратиться из SQL Server, связывая их как связанный сервер, или сослаться на источник данных в функции OPENROWSET или OPENQUERY.

Предложение FROM может также содержать спецификации соединения. Они задают определенный путь для SQL Server, который должен использоваться при навигации от одной таблицы к другой.

Предложение FROM также используется в инструкциях DELETE и UPDATE, чтобы определить таблицы, которые будут изменены.

WHERE search_conditions

Предложение WHERE является фильтром, задающим условия, которым каждая строка в исходных таблицах должна соответствовать, чтобы быть выбранной инструкцией SELECT. Только строки, удовлетворяющие условиям, вносят данные в результирующий набор. Данные из строк, не удовлетворяющих условиям, не используются.

Предложение WHERE также используется в инструкциях DELETE и UPDATE, чтобы определить строки в целевой таблице, которые будут изменены.

GROUP BY group_by_list

Предложение GROUP BY делит результирующий набор на группы на основании значений в столбцах group_by_list. Например, таблица **Sales.SalesOrderHeader** базы данных AdventureWorks имеет десять значений в **TerritoryID**. Предложение GROUP BY **TerritoryID** делит результирующий набор на 10 групп, по одной для каждого значения **TerritoryID.**

HAVING search_conditions

Предложение HAVING является дополнительным фильтром, который применяется к результирующему набору. Логически предложение HAVING фильтрует строки из промежуточного результирующего набора, построенного

путем выполнения любого из предложений FROM, WHERE или GROUP BY в инструкции SELECT. Предложения HAVING обычно используются с предложением GROUP BY, хотя предложение GROUP BY не требуется перед предложением HAVING.

ORDER BY order_list[ASC | DESC]

Предложение ORDER BY определяет порядок, в котором отсортированы строки в результирующем наборе. Параметр order_list указывает столбцы результирующего набора, которые составляют список сортировки. Ключевые слова ASC и DESC используются для указания того, в какой последовательности сортируются строки — возрастающей или убывающей.

Предложение ORDER BY является важным, потому что в соответствии с реляционной теорией нельзя предполагать, что строки в результирующем наборе имеют какую-либо последовательность, если ORDER BY не указано. Предложение ORDER BY должно использоваться в любой инструкции SELECT, для которой важен порядок строк результирующего набора.

Предложения в инструкции SELECT должны быть указаны в соответствующем порядке.

Примеры извлечения столбцов таблицы:

Отобразить все столбцы таблицы Employee

```
USE AdventureWorks2008;
GO
SELECT *
FROM HumanResources.Employee
```

Отобразить данные столбцов FirstName, LastName, JobTitle

```
USE AdventureWorks2008;
GO
SELECT FirstName, LastName, JobTitle
FROM HumanResources.Employee
```

2.2. Фильтрация данных.

Получение конкретных строк в таблице

WHERE использует аргументы для фильтрации данных, запрашиваемых в SELECT. Аргументы поиска не содержат сравнений и критериев отбора данных, выражаются с помощью условных операторов и предикатов. Условные операторы включают такие операторы как: =, <>, <>, <=,> =. Предикаты являются выражениями, которые возвращают TRUE, FALSE или UNKNOWN. К таким выражениям относятся: BETWEEN, CONTAINS, EXISTS, FREETEXT, IN, IS [NOT] NULL, LIKE.. Предикаты используются в условиях поиска пред-

ложений WHERE и HAVING в условиях соединения предложений FROM и других конструкциях, где требуется логическое значение.

Простой запрос с WHERE:

```
USE AdventureWorks2008;
GO
SELECT BusinessEntityID AS ‘Employee Identification Number’, HireDate, VacationHours, SickLeaveHours
FROM HumanResources.Employee
WHERE HireDate > ‘01/01/2000’
```

WHERE с предикатом :

```
USE AdventureWorks2008;
GO
SELECT FirstName, LastName, Phone
FROM Person.Person
WHERE EmailAddress IS NULL;
```

Фильтрация данных с помощью операторов сравнения

Операторы сравнения позволяют проверить, одинаковы ли два выражения. Операторы сравнения можно применять ко всем выражениям, за исключением выражений типов **text**, **ntext** и **image**.

Операторы сравнения используют в предложениях WHERE и HAVING .

Специальный модификатор NOT может быть использован для преобразования логического выражения в обратное.

Пример использования двойного оператора сравнения - используют так же как два выражения сравнения:

```
USE AdventureWorks2008;
GO
SELECT FirstName, LastName, MiddleName
FROM Person.Person
WHERE ModifiedDate >= ‘01/01/2004’
```

Результат:

FirstName	LastName	MiddleName
Ken	Sánchez	J
Terri	Duffy	Lee
Roberto	Tamburello	NULL
Rob	Walters	NULL
Gail	Erickson	A

Фильтрация с помощью сравнения строк

Операторы сравнения поиска строк и подстрок в тексте работают с типами данных text, ntext, char, nchar, varchar, or nvarchar.

Оператор = проверяет точное соответствие между двумя строками.

LIKE определяет, совпадает ли указанная символьная строка с заданным шаблоном. Шаблон может включать обычные символы и символы-шаблоны. Во время сравнения с шаблоном необходимо, чтобы его обычные символы в точности совпадали с символами, указанными в строке. Символы-шаблоны могут совпадать с произвольными элементами символьной строки. Использование символов-шаблонов с оператором LIKE предоставляет больше возможностей, чем использование операторов сравнения строк = и !=. Если тип данных одного из аргументов не является символьной строкой, компонент SQL Server Database Engine, если это возможно, преобразует его в тип данных символьной строки.

FREETEXT предикат, используется в Transact-SQL предложении WHERE. Инструкции Transact-SQL SELECT используются для выполнения полнотекстового поиска по столбцам полнотекстового индекса, содержащим символьные типы данных SQL Server. Этот предикат выполняет поиск значений, которые соответствуют условию поиска по смыслу, а не написанию. Когда используется предикат FREETEXT, ядро полнотекстовых запросов автоматически выполняет описанные далее действия над строкой freetext_string, присваивает вес каждому терму, а затем ищет совпадения.

- Разбивает строку на отдельные слова согласно границам слов (пословное разбиение).
- Формирует словоформы (а также производит выделение основы слова).
- Определяет список расширений или замен для термов на основании совпадений в тезаурусе.

CONTAINS предикат, используемый в предложении WHERE для поиска столбцов, содержащих символьные типы данных, и проверки точного или нечеткого (менее точного) совпадения с отдельными словами и фразами, расстояния между словами или взвешенных совпадений. CONTAINS может производить поиск:

- слова или фразы;
- префикса слова или фразы;
- слова около другого слова;
- слова, флективно сформированного из другого (например, слово «drive» является флективной основой слов «drives», «drove», «driving» и «driven»);
- слова, которое является синонимом другого слова, с использованием тезауруса (например, слово «metal» может иметь синоним «aluminum» и «steel»).

В SQL Server в полнотекстовых предикатах CONTAINS или FREETEXT при выполнении запросов к связанным серверам можно использовать четырехкомпонентные имена.

Примеры возможного использования:

```
WHERE LastName = ‘Johnson’
WHERE LastName LIKE ‘Johns%n’
WHERE CONTAINS(LastName, ‘Johnson’)
WHERE FREETEXT(Description, ‘Johnson’)
```

Результаты:

FirstName	LastName	MiddleName
Abigail	Johnson	NULL
Alexander	Johnson	M
Alexandra	Johnson	J
Alexis	Johnson	J
Alyssa	Johnson	K
Andrew	Johnson	F
Anna	Johnson	NULL

FirstName	LastName	MiddleName
Meredith	Johnsen	NULL
Rebekah	Johnsen	J
Ross	Johnsen	NULL
Willie	Johnsen	NULL
Abigail	Johnson	NULL
Alexander	Johnson	M
Alexandra	Johnson	J

Фильтрация с помощью логических операторов

Логические операторы проверяют истину некоторого условия. Логические операторы, например оператор сравнения, возвращают значение типа **Boolean**: TRUE, FALSE или UNKNOWN.

Оператор	Значение
ALL	TRUE, если все сравнения в наборе равны TRUE.
AND	TRUE, если оба выражения типа Boolean равны TRUE.
ANY	TRUE, если любое из сравнений в наборе равно TRUE.
BETWEEN	TRUE, если операнд принадлежит указанному диапазону.
EXISTS	TRUE, если вложенный запрос возвращает как минимум одну строку.
IN	TRUE, если операнд содержится в заданном списке выражений.
LIKE	TRUE, если оператор удовлетворяет шаблону.

NOT	Меняет значение оператора типа Boolean на противоположное.
OR	TRUE, если одно из выражений типа Boolean равно TRUE.
SOME	TRUE, если некоторые из сравнений в наборе равны TRUE.

При использовании в инструкции нескольких логических операторов первым вычисляется NOT, затем AND и, наконец, OR. Арифметические и побитовые операторы имеют более высокий приоритет, чем логические.

Пример возвращает только строки с first name = ‘John’ и last name = ‘Smith’

```
WHERE FirstName = ‘John’ AND LastName = ‘Smith’
```

Пример возвращает все строки с first name = ‘John’ или last name = ‘Smith’

```
WHERE FirstName = ‘John’ OR LastName = ‘Smith’
```

Пример возвращает все строки с first name =‘John’ и last name не равным ‘Smith’

```
WHERE FirstName = ‘John’ AND NOT LastName = ‘Smith’
```

Получение диапазона значений

BETWEEN - определяет диапазон значений для проверки.

Данные могут быть проверены на диапазон значений. Если данные столбца соответствует одному из значений в диапазоне, строка возвращается в результирующий набор. NOT и может быть использован для обратного результата, строки, возвращаемые в результирующий набор не соответствует диапазону значений.

Если сложное выражение содержит несколько операторов, порядок выполнения этих операторов определяется их приоритетом. Порядок выполнения может существенно повлиять на значение результата.

Уровни приоритета операторов показаны в следующей таблице. Оператор с более высоким уровнем выполняется прежде, чем оператор с более низким уровнем.

Уровень	**Операторы**
1	~ (побитовое НЕ)

2	* (умножение), / (деление), % (остаток от деления)
3	+ (положительное), - (отрицательное), + (сложение), (+ объединение), - (вычитание), & (побитовое И), ^ (побитовое исключающее ИЛИ), | (побитовое ИЛИ)
4	=, >, <, >=, <=, <>, !=, !>, !< (операторы сравнения)
5	NOT
6	AND
7	ALL, ANY, BETWEEN, IN, LIKE, OR, SOME
8	= (присваивание)

Пример возвращает значения данных в пределах заданного диапазона:

```
SELECT OrderDate, AccountNumber, SubTotal, TaxAmt
FROM Sales.SalesOrderHeader
WHERE OrderDate BETWEEN ’08/01/2001’ AND ’08/31/2001’
```

Пример, используется та же логика что и >= AND <=:

```
SELECT OrderDate, AccountNumber, SubTotal, TaxAmt
FROM Sales.SalesOrderHeader
WHERE OrderDate >= ’08/01/2001’
    AND OrderDate <= ’08/31/2001’
```

Результат:

OrderDate	AccountNumber	SubTotal	TaxAmt
2001-08-01 00:00:00.000	10-4020-000018	39677.4848	3174.1988
2001-08-01 00:00:00.000	10-4020-000353	24299.928	1943.9942
2001-08-01 00:00:00.000	10-4020-000206	10295.8366	823.6669
2001-08-01 00:00:00.000	10-4020-000318	1133.2967	90.6637
2001-08-01 00:00:00.000	10-4020-000210	1086.6152	86.9292
2001-08-01 00:00:00.000	10-4020-000164	21923.9352	1753.9148
2001-08-01 00:00:00.000	10-4020-000697	24624.706	1969.9765
2001-08-01 00:00:00.000	10-4020-000191	12286.7218	982.9377

Получение списка значений

IN - Определяет, совпадает ли указанное значение с одним из значений, содержащихся во вложенном запросе или списке.

Пример проверки значений столбцов на соответствие значениям из списка:

```
SELECT SalesOrderID, OrderQty, ProductID, UnitPrice
FROM Sales.SalesOrderDetail
WHERE ProductID IN (750, 753, 765, 770)
```

Пример использует ту же логику что и несколько операторов сравнения OR:

```
SELECT SalesOrderID, OrderQty, ProductID, UnitPrice
FROM Sales.SalesOrderDetail
WHERE ProductID = 750 OR ProductID = 753
    OR ProductID = 765 OR ProductID = 770
```

Результат:

SalesOrderID	OrderQty	ProductID	UnitPrice
43662	5	770	419.4589
43662	3	765	419.4589
43662	1	753	2146.962
43666	1	753	2146.962
43668	2	753	2146.962
43668	6	765	419.4589
43668	2	770	419.4589
43671	1	753	2146.962
43673	2	770	419.4589

2.3. Работа с NULL значениями

Значение NULL указывает на то, что значение неизвестно, оно отличается от пустого или нулевого значения.

Два значения NULL считаются эквивалентными, сравнения между двумя значениями NULL или между значением NULL и каким-либо иным значением возвращают неизвестную величину, так как значение каждого из значений NULL неизвестно. Значение NULL обычно указывает на то, что данные неизвестны, неприменимы или что данные будут добавлены позже. Например, на момент размещения клиентом заказа может быть неизвестно его отчество.

К значениям NULL относятся следующие соглашения:

- для проверки наличия значения NULL в предложении WHERE запроса используются ключевые слова IS NULL или IS NOT NULL;

- при просмотре результатов запроса в редакторе кода среды SQL Server Management Studio значения NULL отображаются в результирующем наборе как NULL;
- значения NULL могут появиться в столбце явным указанием значения NULL в инструкции INSERT или UPDATE, пропуском указания столбца в инструкции INSERT, либо при добавлении нового столбца в существующую таблицу инструкцией ALTER TABLE;
- значения NULL не могут применяться в тех случаях, когда необходимо отличать одну строку таблицы от другой, например в качестве первичного или внешнего ключа.

В программном коде можно реализовать проверку наличия значений NULL так, чтобы определенные вычисления выполнялись только для строк с допустимыми данными, не содержащими значений NULL. Например, в отчете может выводиться столбец социального страхования, если только данные этого столбца не содержат значений NULL. Исключение значений NULL при вычислениях может иметь очень важное значение, так как ряд операций, например подсчет среднего арифметического, могут оказаться неточны, если в этих вычислениях участвуют столбцы, содержащие значения NULL.

Если существует вероятность присутствия в данных значений NULL и это нежелательно, необходимо создавать запросы и инструкции изменения данных так, чтобы они либо удаляли значения NULL, либо преобразовывали их в какие-нибудь другие значения.

Чтобы упростить сопровождение и снизить вероятность возникновения нежелательных последствий для существующих запросов или отчетов, необходимо свести использование значений NULL к минимуму. Планируйте запросы и инструкции изменения данных таким образом, чтобы значения NULL использовались в минимальной степени.

Если в данных присутствуют значения NULL, логические операторы и операторы сравнения могут возвращать, кроме TRUE или FALSE, также и третий результат — UNKNOWN. Необходимость использования трехзначной логики является причиной многих ошибок в приложениях. Приведенные ниже таблицы показывают, как значения NULL влияют на операции сравнения.

Функции работы с NULL значениями

ISNULL - заменяет значение NULL указанным замещающим значением. Определяет, может ли указанное выражение быть NULL.

NOT - задает отрицание логического результата. Предикат меняет возвращаемые выражением значения на обратные, возвращая TRUE, если значение не равно NULL и FALSE, если значение равно NULL.

Для определения, имеет ли выражение значение NULL, используйте IS NULL или IS NOT NULL вместо сравнения операторов (например = или !=). Сравнение операторов возвращает UNKNOWN, если хотя бы один аргумент или они оба равны NULL.

NULLIF - возвращает значение NULL, если два указанных выражения равны. Аналогична поисковому выражению CASE, в котором два выражения равны, а результирующее выражение равно NULL.

В функции NULLIF не рекомендуется использовать такие зависимые от времени функции, как RAND(). Это может приводить к тому, что функция будет вычисляться дважды с возвратом различных результатов для каждого из вызовов.

COALESCE возвращает первое выражение из списка аргументов, не равное NULL. Если все аргументы равны NULL, функция COALESCE возвращает NULL.

Пример с функцией **ISNULL()** возвращает данное значение, если значение столбца NULL:

```
SELECT Description, DiscountPct, MinQty, ISNULL(MaxQty, 0) AS 'Max Quantity'
FROM Sales.SpecialOffer;
```

Пример с функцией **NULLIF()** возвращает NULL если выражения равны:

```
SELECT ProductID, MakeFlag, FinishedGoodsFlag,
  NULLIF(MakeFlag,FinishedGoodsFlag) AS 'Null if Equal'
FROM Production.Product
WHERE ProductID < 10;
```

Пример с функцией **COALESCE()** возвращает первое не NULL выражение из числа аргументов, похоже на работу оператора **CASE:**

```
SELECT CAST(COALESCE(hourly_wage * 40 * 52,  salary,
  commission * num_sales) AS money) AS 'Total Salary'
FROM wages
```

2.4. Форматирование выходных данных

Сортировка данных

ORDER указывает порядок сортировки для столбцов, возвращаемых инструкцией SELECT.

```
 [ ORDER BY
    {  order_by_expression
  [ COLLATE collation_name ]
  [ ASC | DESC ]  } [ ,...n ]
]
```

order_by_expression - указывает столбец, по которому должна выполняться сортировка.

COLLATE {collation_name} - указывает, что операция ORDER BY должна выполняться в соответствии с параметрами сортировки, указанными в аргументе collation_name, но не в соответствии с параметрами сортировки столбца, определенных в таблице или представлении. Значение collation_name может быть именем параметров сортировки Windows или именем параметров сортировки SQL.

ASC - указывает, что значения в указанном столбце должны сортироваться по возрастанию, от меньших значений к большим значениям. Сортировка по умолчанию — ASC.

DESC - указывает, что значения в указанном столбце должны сортироваться по убыванию, от больших значений к меньшим.

Пример:

```
SELECT LastName, FirstName, MiddleName
FROM Person.Person
ORDER BY LastName, FirstName
```

Результат:

LastName	FirstName	MiddleName
Abbas	Syed	E
Abel	Catherine	R.
Abercrombie	Kim	NULL
Abercrombie	Kim	B
Abercrombie	Kim	NULL
Abolrous	Hazem	E
Abolrous	Sam	NULL
Acevedo	Humberto	NULL
Achong	Gustavo	NULL
Ackerman	Pilar	NULL
Ackerman	Pilar	G
Adams	Aaron	B
Adams	Adam	NULL
Adams	Alex	C
Adams	Alexandra	J
Adams	Allison	L
Adams	Amanda	P
Adams	Amber	NULL

Устранение повторяющихся строк

Ключевое слово **DISTINCT** позволяет удалить повторяющиеся строки из результатов, возвращенных инструкцией SELECT. Если ключевое слово DISTINCT не указано, возвращаются все строки, в том числе повторяющиеся.

Пример:

```
SELECT DISTINCT LastName, FirstName, MiddleName
FROM Person.Person
ORDER BY LastName, FirstName
```

Результат:

LastName	FirstName	MiddleName
Abbas	Syed	E
Abel	Catherine	R.
Abercrombie	Kim	NULL
Abercrombie	Kim	B
Abolrous	Hazem	E
Abolrous	Sam	NULL
Acevedo	Humberto	NULL
Achong	Gustavo	NULL
Ackerman	Pilar	NULL
Ackerman	Pilar	G
Adams	Aaron	B
Adams	Adam	NULL
Adams	Alex	C
Adams	Alexandra	J
Adams	Allison	L
Adams	Amanda	P

Именование столбцов результирующего набора

Для имен столбцов можно создать псевдонимы, чтобы облегчить работу с ними, подсчеты и суммирования значений. Например, можно создать псевдоним столбца для:

создания именованного столбца, например «Общий итог», для выражения типа (quantity * unit_price) или агрегатной функции.

создания сокращенной формы имени столбца, например "d_id" для "discounts.stor_id.".

После того, как псевдоним столбца определен, можно использовать его в запросе, чтобы указать данные, выбираемые запросом.

Эквивалентные примеры:

```
SELECT e.BusinessEntityID
 AS ‘Employee Identification Number’
FROM HumanResources.Employee AS e
```

```
SELECT e.BusinessEntityID
 'Employee Identification Number'
FROM HumanResources.Employee e;
```

Результат:

Employee Identification Number
109
4
9
11
158
263
267
270
2
46
106
119
203
269
271
272

Использование строковых литералов

SUBSTRING возвращает фрагмент символьного, двоичного, текстового или графического выражения.

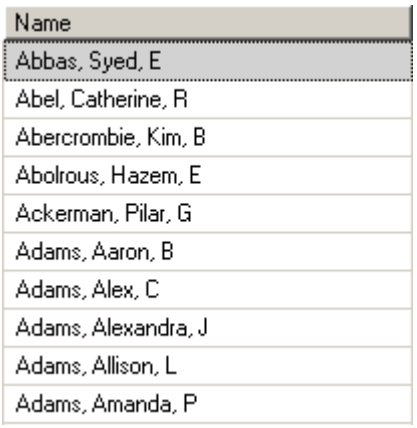
Строковые литералы:

- Являются константами.
- Могут быть вставлены в производные столбцы в формате данных.
- Могут использоваться в качестве альтернативных значений в таких функциях как ISNULL().

Пример:

```
SELECT (LastName + ', ' + FirstName + ' ' + ISNULL(SUBSTRING(MiddleName, 1, 1), ' ')) AS Name
FROM Person.Person
ORDER BY LastName, FirstName, MiddleName
```

Результат:



Name
Abbas, Syed, E
Abel, Catherine, R
Abercrombie, Kim, B
Abolrous, Hazem, E
Ackerman, Pilar, G
Adams, Aaron, B
Adams, Alex, C
Adams, Alexandra, J
Adams, Allison, L
Adams, Amanda, P

Использование выражений

Выражение - сочетание символов и операторов, используемое компонентом SQL Server Database Engine для вычисления одиночного значения данных. Отдельные константы, переменные, столбцы и скалярные функции являются примерами простых выражений. Для соединения двух и более простых выражений в одно сложное можно использовать операторы.

Два выражения можно объединить каким-либо оператором, если оба они содержат данные типов, поддерживаемых оператором, и выполняется хотя бы одно из следующих условий.

Выражения содержат данные одинаковых типов.

Тип данных с более низким приоритетом может быть неявно преобразован в тип данных с более высоким приоритетом.

Если выражения не удовлетворяют этим условиям, функция CAST или CONVERT явным образом преобразует тип данных с более низким приоритетом или в тип данных с более высоким приоритетом, или в промежуточный тип данных, который затем может быть неявно преобразован в тип данных с более высоким приоритетом.

Если неявное или явное преобразование не поддерживается, эти два выражения объединить невозможно.

Параметры сортировки любого выражения, результатом которого является символьная строка, определяются правилами очередности параметров сортировки.

Пример использования математических выражений в SELECT и предложении WHERE:

```
SELECT Name, ProductNumber, ListPrice AS OldPrice, (ListPrice * 1.1) AS NewPrice
FROM Production.Product
WHERE ListPrice > 0 AND (ListPrice/StandardCost) > .8
```

Пример использования функций:

```
SELECT Name, ProductNumber, ListPrice AS OldPrice, (ListPrice * 1.1) AS NewPrice
FROM Production.Product
WHERE SellEndDate < GETDATE()
```

2.5. Вопросы производительности в написании запросов

При создании эффективных запросов следует избегать:

- Масок– **LIKE '%an'**
- Отрицательных результатов проверок …=, **NOT IN('California')**
- Выражений которые используют два столбца одной таблицы
- Выражений, которые используют имена столбцов при поисках аргументов – **WHERE (Price + $10) > $20**

При создании эффективных запросов следует помнить:

- Шаблоны, размещенные в середине или конце выражения, занимают меньше ресурсов
- По возможности использовать положительные результаты проверок
- Экономно использовать расчеты при поиске аргументов
- По возможности использовать индексированные столбцы при поиске аргументов

Оптимизировать запросы помогут несколько правил:

- При использовании встроенных функций по возможности использовать постоянные значения аргументов
- Строить индексы, которые будут использоваться при поиске аргументов
- Добавить памяти к серверам
- Использовать несколько процессоров на сервере
- Не использовать несколько псевдонимов для одной таблицы в запросе

3. Группировка и обобщение данных

3.1. Суммирование данных с помощью агрегатных функций

Статистические функции выполняют вычисление на наборе значений и возвращают одиночное значение. Статистические функции, за исключением COUNT, не учитывают значения NULL. Статистические функции часто используются в выражении GROUP BY инструкции SELECT.

Все статистические функции являются детерминированными. Это означает, что статистические функции возвращают одну и ту же величину при каждом их вызове на одном и том же наборе входных значений.

Предложение OVER может следовать за всеми статистическими функциями, кроме CHECKSUM.

Статистические функции могут быть использованы в качестве выражений только в следующих случаях.

- Список выбора инструкции SELECT (вложенный или внешний запрос).
- Предложение COMPUTE или COMPUTE BY.
- Предложение HAVING.

Transact-SQL предоставляет следующие статистические функции.

AVG, MIN, CHECKSUM, AGG, SUM, COUNT, STDEV, COUNT_BIG, STDEVP, GROUPING, VAR, MAX, VARP

Пример использования встроенной агрегатной функции:

```
USE AdventureWorks
SELECT MAX(TaxRate)
FROM Sales.SalesTaxRate
GROUP BY TaxType;
```

Использование агрегатных функций с NULL

Возвращает количество элементов в группе. Функция COUNT работает как функция COUNT_BIG. Единственное различие между двумя функциями — возвращаемые значения. Функция COUNT всегда возвращает значение типа **int**. Функция COUNT_BIG всегда возвращает значение типа данных **bigint**. Может следовать за предложением OVER.

Функция COUNT(*) возвращает количество элементов в группе. Сюда входят значения NULL и повторяющиеся значения.

Функция COUNT(ALL expression) оценивает expression для каждой строки в группе и возвращает количество значений, не равных NULL.

Функция COUNT(DISTINCT expression) оценивает expression для каждой строки в группе и возвращает количество уникальных значений, не равных NULL.

Для возвращаемых значений, больших 2^31-1, функция COUNT формирует сообщение об ошибке. Вместо этого следует использовать COUNT_BIG.

Пример:

```
USE AdventureWorks
SELECT AVG(ISNULL(Weight,0)) AS ‘AvgWeight’
FROM Production.Product
```

Интеграция сборки CLR

Начиная с версии SQL Server 2005, SQL Server обеспечивает интеграцию с компонентами CLR платформы .NET Framework для Microsoft Windows. Это означает, что хранимые процедуры, триггеры, определяемые пользователем, типы, определяемые пользователем функций, определяемые пользователем статистические функции и возвращающие табличные значения потоковых функций теперь могут разрабатываться с использованием любого языка .NET Framework, включая Microsoft Visual Basic .NET и Microsoft Visual C#.

Пространство имен **Microsoft.SqlServer.Server** содержит основные возможности программирования CLR для SQL Server. Пространство имен **Microsoft.SqlServer.Server** документировано в пакете .NET Framework SDK.

Сборки являются файлами динамической библиотеки, которые используются в экземпляре SQL Server для развертывания функций, хранимых процедур, триггеров, определяемых пользователем статистических вычислений и определяемых пользователем типов, записанных на одном из языков управляемого кода, содержащимся в среде CLR Microsoft.NET Framework, а не на языке Transact-SQL.

Сборка в SQL Server представляет собой объект, который ссылается на управляемый модуль приложений (DLL-файл), созданный в среде CLR .NET Framework. Сборка содержит метаданные класса и управляемый код. Передача сборки на экземпляр SQL Server — это первый шаг к созданию любого из следующих объектов базы данных.

В SQL Server сборки выполняют следующие функции:

- Содержат управляемый код, который выполняет функциональность одного или нескольких объектов базы данных среды CLR, перечисленных ранее.
- Содержат метаданные, которые включают в себя номер версии и культуру сборки, дополнительный открытый ключ, который уникально идентифицирует список классов сборки, методы, определенные в сборке, и архитектуру процессора сборки.
- Управляют степенью, к которой управляемый код может получить доступ внешних ресурсов с помощью регулирования разрешений кода доступа.
- Содержат метаданные о зависимостях от других сборок, на которые ссылается сборка.

Пользовательские статистические функции среды CLR

CREATE AGGREGATE - создает определяемую пользователем статистическую функцию, реализация которой определена в классе сборки платформы .NET Framework. Чтобы компонент Database Engine привязал статистическую функцию к ее реализации, содержащая реализацию сборка платформы

.NET Framework должна быть сначала передана в экземпляр SQL Server с помощью инструкции CREATE ASSEMBLY.

Класс сборки, указанной в аргументе assembly_name, и его методы должны соответствовать всем требованиям к реализации определяемых пользователем статистических функций в экземпляре SQL Server.

Пример:

```
CREATE ASSEMBLY StringUtilities FROM ‘PathToAssembly\StringUtilities.dll'
WITH PERMISSION_SET=SAFE;
```

```
CREATE AGGREGATE Concatenate(@input nvarchar(4000))
RETURNS nvarchar(4000)
EXTERNAL NAME [StringUtilities].[Concatenate]
```

Статистические функции выполняют вычисление на наборе значений и возвращают одиночное значение. Традиционно MicrosoftSQL Server поддерживал только встроенные статистические функции, например **SUM** и **MAX**, оперирующие с набором входных скалярных величин и создающие из этого набора единый статистический показатель. Но теперь интеграция SQL Server со средой Microsoft .NET Framework CLR позволяет создавать в управляемом коде пользовательские статистические функции и предоставлять доступ к ним из Transact-SQL или другого управляемого кода.

Вызов определяемых пользователем статистических функций CLR

В инструкциях Transact-SQL SELECT можно вызывать пользовательские статистические функции CLR, на которые распространяются те же правила, что и на системные статистические функции.

Применяются следующие дополнительные правила:

- Текущий пользователь должен иметь разрешение **EXECUTE** на пользовательскую статистическую функцию.
- Пользовательские статистические функции можно вызывать с помощью двусоставного имени в следующей форме: schema_name.udagg_name.
- Тип аргумента пользовательской статистической функции должен совпадать с input_type статистической функции, как определено в инструкции **CREATE AGGREGATE**, или неявно преобразовываться в него.
- Тип данных, возвращаемых пользовательской статистической функцией, должен совпадать с return_type в инструкции **CREATE AGGREGATE**.

3.2. Суммирование сгруппированных данных

Использование GROUP BY

GROUP BY - группирует выбранный набор строк для получения набора сводных строк по значениям одного или нескольких столбцов или выражений. Возвращается одна строка для каждой группы. Статистические функции в списке <select> предложения SELECT предоставляют информацию о каждой группе(Рис.4.), а не об отдельных строках.

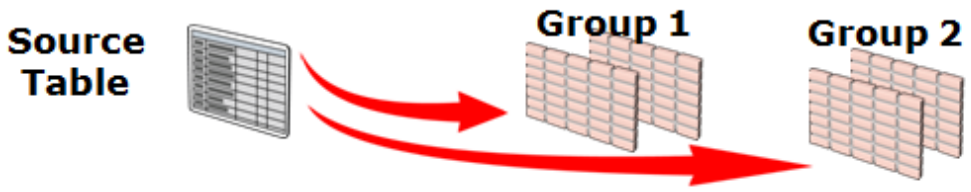


Рис.4. Группировка данных.

Предложение GROUP BY имеет два синтаксиса: совместимый с ISO и несовместимый с ISO. В каждой отдельной инструкции SELECT может использоваться только один стиль синтаксиса. Во всех новых разработках используйте совместимый с ISO синтаксис. Синтаксис, несовместимый с ISO, служит для обеспечения обратной совместимости.

В этом разделе предложение GROUP BY можно описать как общее или простое.

- Общее предложение GROUP BY включает конструкции GROUPING SETS, CUBE, ROLLUP, WITH CUBE и WITH ROLLUP.
- Простое предложение GROUP BY не включает конструкции GROUPING SETS, CUBE, ROLLUP, WITH CUBE и WITH ROLLUP. Предложение GROUP BY (), предназначенное для определения общего итога, рассматривается как простое предложение GROUP BY.

Пример:

```
SELECT SalesOrderID, SUM(LineTotal) AS SubTotal
FROM Sales.SalesOrderDetail
GROUP BY SalesOrderID
ORDER BY SalesOrderID
```

Результат:

SalesOrderID	SubTotal
1	23761
2	45791
3	75909
4	19900

Фильтрация сгруппированных данных с помощью HAVING

HAVING - определяет условие поиска для группы или статистического выражения. Предложение HAVING можно использовать только в инструкции SELECT. Предложение HAVING обычно используется в предложении GROUP BY. Когда GROUP BY не используется, предложение HAVING работает так же, как и предложение WHERE.

Пример:

```
SELECT SalesOrderID, SUM(LineTotal) AS SubTotal
FROM Sales.SalesOrderDetail
GROUP BY SalesOrderID
HAVING SUM(LineTotal) > 100000.00
ORDER BY SalesOrderID
```

Результат:

SalesOrderID	SubTotal
43875	121761.939600
43884	115696.331324
44518	126198.336168
44528	108783.587200
44530	104958.806836
44795	104111.515642
46066	100378.907800
...	

Суммирование сгруппированных данных GROUP BY

GROUP BY:

• обеспечивает представленные наборы данных для статистических функции,

• при использовании функции агрегации, столбцы в списке выбора должны быть либо в функции или в GROUP BY,

• обратите внимание, что в в объединенной таблице колонки сгруппированы.

HAVING:

• может использоваться вместо WHERE,

• фильтры группировки данных используются после того, как группировка произошло, но перед выводом в качестве результатов.

Пример:

```
SELECT SalesOrderID, SUM(LineTotal) AS SubTotal
FROM Sales.SalesOrderDetail
GROUP BY SalesOrderID
HAVING SUM(LineTotal) > 100000.00
ORDER BY SalesOrderID
```

Результат:

SalesOrderID	SubTotal
43875	121761.939600
43884	115696.331324
44518	126198.336168
44528	108783.587200
44530	104958.806836
44795	104111.515642
46066	100378.907800
...	

ROLLUP и CUBE

Общее предложение GROUP BY включает конструкции GROUPING SETS, CUBE, ROLLUP, WITH CUBE и WITH ROLLUP.

ROLLUP - формирует статистические строки простого предложения GROUP BY и строки подытогов или строки со статистическими вычислениями высокого уровня, а также строки общего итога.

Количество возвращаемых группирований равно количеству выражений в <составном списке элементов> плюс один. Для каждого уникального сочетания значений формируется одна строка с подытогом. Вычисляется также строка общего итога. Столбцы складываются справа налево. Последовательность расположения столбцов влияет на выходное группирование ROLLUP и может отразиться на количестве строк в результирующем наборе.

Пример:

```
SELECT ProductID,Shelf,SUM(Quantity) AS QtySum
FROM Production.ProductInventory
WHERE ProductID<6
GROUP BY ROLLUP(ProductID,Shelf)
```

Результат:

ProductID	Shelf	QtySum
1	A	761
1	B	324
1	NULL	1085
2	A	791
2	B	318
2	NULL	1109
3	A	909
3	B	443
3	NULL	1352
4	A	900

CUBE - формирует статистические строки простого предложения GROUP BY, строки со статистическими вычислениями высокого уровня конструкции ROLLUP и строки с результатами перекрестных вычислений.

Выходные данные CUBE являются группированием для всех перестановок выражений в <составном списке элементов>.

Количество формируемых группирований равно (2^n), где n — количество выражений в <составном списке элементов>. Формируется одна строка для

каждого уникального сочетания значений с подытогом для каждой строки и строкой общего итога.

Выходные данные CUBE не зависят от порядка столбцов.

Пример:

```
SELECT ProductID,Shelf,SUM(Quantity) AS QtySum
FROM Production.ProductInventory
WHERE ProductID<6
GROUP BY CUBE(ProductID, Shelf)
```

Результат:

ProductID	Shelf	QtySum
1	A	761
2	A	791
3	A	909
4	A	900
NULL	A	3361
1	B	324
2	B	318
3	B	443
4	B	442
NULL	B	1507

Суммирование сгруппированных данных COMPUTE и COMPUTE BY

COMPUTE - формирует итоги, которые появляются в дополнительном столбце сводки в конце результирующего набора. При использовании с ключевым словом BY предложение COMPUTE формирует в результирующем наборе сегменты и промежуточные итоги. В одном запросе можно указать одновременно COMPUTE BY и COMPUTE.

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

Первый пример выдает общие итоги для SalesOrderDetail:

```
SELECT SalesOrderID, UnitPrice, UnitPriceDiscount
FROM Sales.SalesOrderDetail
ORDER BY SalesOrderID
COMPUTE SUM(UnitPrice), SUM(UnitPriceDiscount)
```

Результат:

SalesOrderID	UnitPrice	UnitPriceDiscount
1	50	200
2	135	350
3	NULL	1085

sum	Sum
185	1635

Второй пример дает общий итог расчетов, а также промежуточные итоги разграничены по SalesPersonID. Поэтому во втором примере - два набора результатов для каждого SalesPersonID и полный набор результатов, аналогичный первому примеру:

```
SELECT SalesPersonID, CustomerID, OrderDate, SubTotal, TotalDue
FROM Sales.SalesOrderHeader
ORDER BY SalesPersonID, OrderDate
COMPUTE SUM(SubTotal), SUM(TotalDue) BY SalesPersonID
```

GROUPING SETS

Предложение GROUP BY с оператором GROUPING SETS может дать такой же результирующий набор, как и несколько простых предложений GROUP BY, объединенных с помощью UNION ALL. Оператор GROUPING SETS может дать результат, эквивалентный результату простой операции GROUP BY, операции ROLLUP или CUBE. Разные сочетания операций GROUPING SETS, ROLLUP или CUBE могут создавать эквивалентные результирующие наборы.

Пример:

```
SELECT T.[Group] AS 'Region', T.CountryRegionCode AS 'Country'
   ,S.Name AS 'Store', H.SalesPersonID
   ,SUM(TotalDue) AS 'Total Sales'
FROM Sales.Customer C
```

```
   INNER JOIN Sales.Store S
      ON C.StoreID  = S.BusinessEntityID
   INNER JOIN Sales.SalesTerritory T
      ON C.TerritoryID  = T.TerritoryID
   INNER JOIN Sales.SalesOrderHeader H
      ON C.CustomerID = H.CustomerID
WHERE T.[Group] = 'Europe'
   AND T.CountryRegionCode IN('DE', 'FR')
   AND SUBSTRING(S.Name,1,4)IN('Vers', 'Spa ')
GROUP BY GROUPING SETS
   (T.[Group], T.CountryRegionCode, S.Name, H.SalesPersonID)
ORDER BY T.[Group], T.CountryRegionCode, S.Name, H.SalesPersonID;
```

Результат:

Region	**Country**	**Store**
NULL	NULL	NULL
NULL	NULL	NULL
NULL	NULL	NULL
NULL	NULL	Spa and Exercise Outfitters
NULL	NULL	Versatile Sporting Good Company
NULL	DE	NULL
NULL	FR	NULL
Europe	NULL	NULL

3.3. Рейтинг сгруппированных данных

Ранжирование

Ранжирующие функции возвращают ранжирующее значение для каждой строки в секции. В зависимости от используемой функции значения некоторых строк могут совпадать. Ранжирующие функции являются недетерминированными.

Transact-SQL содержит следующие ранжирующие функции: RANK, NTILE , DENSE_RANK, ROW_NUMBER .

OVER определяет секционирование и упорядочение набора строк до применения соответствующей оконной функции.

Оконные функции определены в стандарте ISO SQL. В SQL Server предоставляются ранжирующие и статистические оконные функции. Окно — это набор строк, определяемый пользователем. Оконная функция вычисляет значение для каждой строки в результирующем наборе, полученном из окна.

В одном запросе с одним предложением FROM может использоваться несколько статистических или ранжирующих оконных функций. Однако предложение OVER для каждой функции может использовать свое секционирование и упорядочение. Предложение OVER не может использоваться со статистической функцией CHECKSUM.

Ранжирование с помощью RANK, DENSE_RANK, ROW_NUMBER , NTILE.

RANK - возвращает ранг каждой строки в секции результирующего набора. Ранг строки вычисляется как единица плюс количество рангов, находящихся до этой строки.

Если две и более строки претендуют на один ранг, то все они получат одинаковый ранг. Порядок сортировки, используемый для всего запроса, определяет порядок, в котором строки будут появляться в результирующем наборе.

Пример:

```
SELECT P.Name Product, P.ListPrice, PSC.Name Category,
RANK() OVER(PARTITION BY PSC.Name
ORDER BY P.ListPrice DESC)
AS PriceRank
FROM Production.Product P
JOIN Production.ProductSubCategory PSC
ON P.ProductSubCategoryID = PSC.ProductSubCategoryID
```

Результат:

Product	**ListPrice**	**Category**	**PriceRank**
Men's Bib-Shorts, S	89.99	Bib-Shorts	1
Men's Bib-Shorts, M	89.99	Bib-Shorts	1
Men's Bib-Shorts, L	89.99	Bib-Shorts	1
Hitch Rack - 4-Bike	120.00	Bike Racks	1
All-Purpose Bike Stand	159.00	Bike Stands	1
Mountain Bottle Cage	9.99	Bottles and Cages	1
Road Bottle Cage	8.99	Bottles and Cages	2

DENSE_RANK - возвращает ранг строк в секции результирующего набора без промежутков в ранжировании. Ранг строки равен количеству различных значений рангов, предшествующих строке, увеличенному на единицу.

Если две или более строк одной секции равны при ранжировании, каждой такой строке присваивается один и тот же ранг. Порядок сортировки строк в результате определяется порядком сортировки результата всего запроса. Из этого следует, что строка с рангом 1 не всегда является первой строкой в секции.

Пример:

```
SELECT P.Name Product, P.ListPrice, PSC.Name Category,
DENSE_RANK()
OVER(PARTITION BY PSC.Name ORDER BY P.ListPrice DESC)
AS PriceRank
FROM Production.Product P
JOIN Production.ProductSubCategory PSC
ON P.ProductSubCategoryID = PSC.ProductSubCategoryID
```

Результат:

Product	**ListPrice**	**Category**	**PriceRank**
HL Mountain Handlebars	120.27	Handlebars	1
HL Road Handlebars	120.27	Handlebars	1
HL Touring Handlebars	91.57	Handlebars	2
ML Mountain Handlebars	61.92	Handlebars	3
HL Headset	124.73	Headsets	1
ML Headset	102.29	Headsets	2
LL Headset	34.20	Headsets	3

ROW_NUMBER - возвращает последовательный номер строки в секции результирующего набора, 1 соответствует первой строке в каждой из секций.

Предложение ORDER BY определяет последовательность, в которой строкам назначаются уникальные номера с помощью функции ROW_NUMBER в пределах указанной секции

Пример:

```
SELECT ROW_NUMBER()
OVER(PARTITION BY PC.Name ORDER BY ListPrice)
AS Row, PC.Name Category, P.Name Product, P.ListPrice
FROM Production.Product P
JOIN Production.ProductSubCategory PSC
ON P.ProductSubCategoryID = PSC.ProductSubCategoryID
JOIN Production.ProductCategory PC
ON PSC.ProductCategoryID = PC.ProductCategoryID
```

Результат:

Row	Category	Product	ListPrice
1	Accessories	Patch Kit/8 Patches	2.29
2	Accessories	Road Tire Tube	3.99
1	Bikes	Road-750 Black, 44	539.99
2	Bikes	Road-750 Black, 44	539.99

NTILE - Распределяет строки упорядоченной секции в заданное количество групп. Группы нумеруются, начиная с единицы. Для каждой строки функция NTILE возвращает номер группы, которой принадлежит строка.

Если количество строк в секции не делится на integer_expression, формируются группы двух размеров, отличающихся на единицу. В порядке, заданном предложением OVER, группы большего размера следуют перед группами меньшего размера. Например, если общее число строк равно 53, а число групп равно пяти, первые три группы будут состоять из 11 строк, а оставшиеся две — из 10. С другой стороны, если общее число строк делится на число групп, строки распределяются равномерно по всем группам. Например, если общее число строк равно 50 и задано пять групп, каждый сегмент будет состоять из 10 строк.

Пример:

```
SELECT NTILE(3) OVER(PARTITION BY PC.Name ORDER BY ListPrice)
AS PriceBand, PC.Name Category, P.Name Product, P.ListPrice
FROM Production.Product P
JOIN Production.ProductSubCategory PSC
ON P.ProductSubCategoryID = PSC.ProductSubCategoryID
JOIN Production.ProductCategory PC
ON PSC.ProductCategoryID = PC.ProductCategoryID
```

Результат:

PriceBand	Category	Product	ListPrice
1	Accessories	Patch Kit/8 Patches	2.29
1	Accessories	Road Tire Tube	3.99
2	Accessories	LL Road Tire	21.49
2	Accessories	ML Road Tire	24.99
3	Accessories	Sport-100 Helmet, Blue	34.99
3	Accessories	HL Mountain Tire	35.00
1	Bikes	Road-750 Black, 44	539.99
1	Bikes	Road-650 Red, 48	782.99
2	Bikes	Road-650 Red, 52	782.99

Классификация функций ранжирования

Возможности	RANK	DENSE_RANK	NTILE	ROW_NUMBER
Ряды равномерно распределены между группами			+	
Строкам одного ранга назначается одно и то же значение	+	+		
Последовательная нумерация строк в разделе				+

Используя таблицу классификации можно выделить опреленные бизнес – сценарии для наиболее удачного использования функций ранжирования:

NTILE:

хранилища данных,

разделение команд продаж в равные по размеру группы на основании показателей продаж,

RANK, DENSE_RANK

рейтинг лучших продаж по дням в году,

рейтинг самых продаваемых продуктов,

ROW_NUMBER

количество контактов с пользователями при заказе техники.

3.4. Создание перекрестных запросов

PIVOT и UNPIVOT

Реляционные операторы PIVOT и UNPIVOT можно использовать для изменения возвращающего табличное значение выражения в другой таблице. Оператор PIVOT разворачивает возвращающее табличное значение выражение, преобразуя уникальные значения одного столбца выражения в несколько выходных столбцов, а также, в случае необходимости, объединяет оставшиеся повторяющиеся значения столбца и отображает их в выходных данных. Опера-

тор UNPIVOT производит действия, обратные PIVOT, преобразуя столбцы возвращающего табличное значение выражения в значения столбца.

Оператор UNPIVOT не является в точности обратным оператору PIVOT. Оператор PIVOT производит статистическую обработку и слияние нескольких строк в единую выходную строку. Оператор UNPIVOT не восстанавливает исходные возвращающие табличное значение выражения, так как строки были объединены. Оператор UNPIVOT удаляет пустые значения из обрабатываемых им данных. Поэтому в случае наличия в исходных столбцах пустых значений данные на выходе оператора UNPIVOT могут отличаться от данных до их обработки с помощью оператора PIVOT.

Пример:

PIVOT – конвертирует сводные значения в таблицу

Результат:

Cust	Prod	Qty
Mike	Bike	3
Mike	Chain	2
Mike	Bike	5
Lisa	Bike	3
Lisa	Chain	3
Lisa	Chain	4

Cust	Bike	Chain
Mike	8	2
Lisa	3	7

Пример:

UNPIVOT – конвертирует столбцы в данные

Cust	Bike	Chain
Mike	8	2
Lisa	3	7

Cust	Prod	Qty
Mike	Bike	8
Mike	Chain	2
Lisa	Bike	3
Lisa	Chain	7

Пример, показывает использование оператора PIVOT для создания перекрестного запроса. В этом примере запрашивается таблица заказов для определения количества продукции, размещенной для определенных клиентов.

Cust	Prod	Qty
Mike	Bike	3
Mike	Chain	2
Mike	Bike	5
Lisa	Bike	3
Lisa	Chain	3
Lisa	Chain	4

```
SELECT * FROM
Sales.Order
PIVOT (SUM(Qty) FOR Prod
IN ([Bike],[Chain])) PVT
```

Cust	Bike	Chain
Mike	8	2
Lisa	3	7

Пример показывает использование UNPIVOT для изменения наборов в результирующих наборах данных.

Cust	Bike	Chain
Mike	8	2
Lisa	3	7

```
SELECT Cust, Prod, Qty
FROM Sales.PivotedOrder
UNPIVOT (Qty FOR Prod IN
([Bike],[Chain])) UnPVT
```

Cust	Prod	Qty
Mike	Bike	8
Mike	Chain	2
Lisa	Bike	3
Lisa	Chain	7

4. Объединение данных из нескольких таблиц.

4.1. Основы объединений. Виды предложений с типами объединений

Условия соединения можно задать в предложении FROM или WHERE. Рекомендуется указывать их в предложении FROM. Предложения WHERE и HAVING могут также содержать условия поиска для дальнейшей фильтрации строк, выбранных этими условиями соединения.

Соединения можно разделить на следующие категории.

- Внутренние соединения (типичные операции соединения, использующие такие операторы сравнения, как = или <>). Они включают эквивалентные соединения и естественные соединения.

Внутренние соединения используют оператор сравнения для установки соответствия строк из двух таблиц на основе значений общих столбцов в каждой таблице. Примером может быть получение всех строк, в которых идентификационный номер студента одинаковый как в таблице **students**, так и в таблице **courses**.

- Внешние соединения. Внешние соединения бывают левыми, правыми и полными.

Если внешние соединения задаются в предложении FROM, они указываются с одним из следующих наборов ключевых слов.

- LEFT JOIN или LEFT OUTER JOIN

Результирующий набор левого внешнего соединения включает все строки из левой таблицы, заданной в предложении LEFT OUTER, а не только те, в которых соединяемые столбцы соответствуют друг другу. Если строка в левой таблице не имеет совпадающей строки в правой таблице, результирующий набор строк содержит значения NULL для всех столбцов списка выбора из правой таблицы.

- RIGHT JOIN или RIGHT OUTER JOIN

Правое внешнее соединение является обратным для левого внешнего соединения. Возвращаются все строки правой таблицы. Для левой таблицы возвращаются значения NULL каждый раз, когда строка правой таблицы не имеет совпадающей строки в левой таблице.

- FULL JOIN или FULL OUTER JOIN

Полное внешнее соединение возвращает все строки из правой и левой таблицы. Каждый раз, когда строка не имеет соответствия в другой таблице, столбцы списка выбора другой таблицы содержат значения NULL. Если между таблицами имеется соответствие, вся строка результирующего набора содержит значения данных из базовых таблиц.

- Перекрестные с соединения

Перекрестное соединение возвращает все строки из левой таблицы. Каждая строка из левой таблицы соединяется со всеми строками из правой таблицы. Перекрестные соединения называются также декартовым произведением.

Inner Join

Внутреннее соединение — это такое соединение, в котором значения в соединяемых столбцах подвергаются сравнению с использованием оператора сравнения.

В стандарте SQL внутренние соединения задаются предложением FROM или предложением WHERE. Это единственный тип соединения, допустимый в стандарте SQL для предложения WHERE. Внутренние соединения, задаваемые предложением WHERE, известны как внутренние соединения старого типа.

Такое внутреннее соединение называется эквисоединением. При таком соединении сохраняются все столбцы обеих таблиц и только те строки, для которых в соединяющем столбце имеется равное значение.

Пример:

```
SELECT e.LoginID
FROM HumanResources.Employee AS e
   INNER JOIN Sales.SalesPerson AS s
   ON e.BusinessEntityID = s.BusinessEntityID
```

Результат:

```
LoginID
------------------------
adventure-works\syed0
```

```
adventure-works\david8
adventure-works\garrett1
...
(17 row(s) affected)
```

Outer Join

Внутренние соединения возвращают результат, когда в обеих таблицах есть хотя бы одна строка, соответствующая условиям соединения. Внутренние соединения исключают строки, не соответствующие ни одной строке в другой таблице. Однако внешние соединения возвращают все строки хотя бы из одной таблицы или представления, упомянутые в предложении FROM, если они удовлетворяют условиям поиска WHERE или HAVING. Все строки, получаемые из левой таблицы, образуют левое внешнее соединение, а строки, получаемые из правой таблицы, — правое внешнее соединение. Все строки их обеих таблиц возвращаются в полном внешнем соединении.

Для внешних соединений в предложении FROM SQL Server использует ключевые слова ISO:

- LEFT OUTER JOIN или LEFT JOIN;
- RIGHT OUTER JOIN или RIGHT JOIN;
- FULL OUTER JOIN или FULL JOIN.

Пример:

```
SELECT p.Name, pr.ProductReviewID
FROM Production.Product p
LEFT OUTER JOIN Production.ProductReview pr
ON p.ProductID = pr.ProductID
```

Результат:

```
Name            ProductReviewID
--------------------------------
Adjustable Race   NULL
Bearing Ball      NULL
...
(505 row(s) affected)
```

Cross Join

Перекрестное соединение, не имеющее предложения WHERE, выполняет декартово произведение таблиц, вовлеченных в объединение. Размер результирующего набора декартова произведения вычисляется как произведение количества строк в первой таблице на количество строк во второй таблице.

Пример:

```
SELECT p.BusinessEntityID, t.Name AS Territory
FROM Sales.SalesPerson p
CROSS JOIN Sales.SalesTerritory t
ORDER BY p.BusinessEntityID
```

Результат:

```
BusinessEntityID  Territory
--------------------------
274               Northwest
274               Northeast
...
(170 row(s) affected)
```

Декартово произведение

Декартово произведение, или перекрестное соединение, определяется как все возможные комбинации строк во всех таблицах. Возврат декартова произведения на таблицах с большим количеством записей и / или из множества таблиц может занять несколько часов.

4.2.Применение объединений в отчетах

Соединение трех и более таблиц

Хотя в операции соединения указываются всего две таблицы, предложение FROM может содержать несколько операций объединения. Это позволяет соединять в одном запросе несколько таблиц.

Таблица ProductVendor базы данных AdventureWorks является хорошим примером ситуации, в которой может понадобиться соединение более двух таблиц.

Пример:

```
SELECT p.Name, v.Name
FROM Production.Product p
JOIN Purchasing.ProductVendor pv
ON p.ProductID = pv.ProductID
JOIN Purchasing.Vendor v
ON pv.BusinesEntityID = v.BusinessEntityID
WHERE ProductSubcategoryID = 15
ORDER BY v.Name
```

Результат:

```
Name                    Name
------------------------------------------
LL Mountain Seat/Saddle   Chicago City Saddles
ML Mountain Seat/Saddle   Chicago City Saddles
...
(18 row(s) affected)
```

Самообъединение

Таблицу можно соединить с собой в самосоединение. Используйте самосоединение, когда требуется создать результирующий набор, который соединяет записи в таблице с другими записями в той же таблице. Чтобы указать таблицу два раза в одном и том же запросе, необходимо задать псевдоним таблицы хотя бы для одного экземпляра имени таблицы. Этот псевдоним таблицы помогает обработчику запросов определить, какие данные должны быть представлены в столбцах (из правой или из левой версии таблицы).

Пример:

```
SELECT DISTINCT pv1.ProductID, pv1.BusinessEntityID
FROM Purchasing.ProductVendor pv1
INNER JOIN Purchasing.ProductVendor pv2
ON pv1.ProductID = pv2.ProductID
   AND pv1.BusinessEntityID <> pv2.BusinessEntityID
ORDER BY pv1.ProductID
```

Результат:

```
ProductID   BusinessEntityID
-----------------------------
317         1578
317         1678
...
(347 row(s) affected)
```

Объединение таблиц с помощью неравенства

Внутреннее соединение — это такое соединение, в котором значения в соединяемых столбцах подвергаются сравнению с использованием оператора сравнения.

В стандарте SQL внутренние соединения задаются предложением FROM или предложением WHERE. Это единственный тип соединения, допустимый в стандарте SQL для предложения WHERE. Внутренние соединения, задаваемые предложением WHERE, известны как внутренние соединения старого типа.

Имеется также возможность соединения неравных значений двух столбцов. Чтобы соединить неравные значения, используются те же операторы и предикаты, что и для внутренних соединений.

Пример:

```
SELECT DISTINCT p1.ProductSubcategoryID, p1.ListPrice
FROM Production.Product p1
   INNER JOIN Production.Product p2
   ON p1.ProductSubcateogoryID = p2.ProductSubcategoryID
   AND p1.ListPrice <> p2.ListPrice
WHERE p1.ListPrice < $15 AND p2.ListPrice < $15
ORDER BY ProductSubcategoryID
```

Результат:

```
ProductSubcateogoryID    ListPrice
---------------------------------
23                       8.99
23                       9.50
...
(8 row(s) affected)
```

Объединение таблиц в пользовательской функции

Определяемые пользователем встроенные функции представляют собой подмножество определяемых пользователем функций, возвращающих значения типа данных table. Встроенные функции могут быть использованы для улучшения работы параметризованных представлений.

Определяемые пользователем встроенные функции используют следующие правила.

- Предложение RETURNS содержит только ключевое слово table. Нет необходимости задавать формат возвращаемой переменной, поскольку он определяется форматом результирующего набора инструкции SELECT в предложении RETURN.
- Нет инструкции function_body, разделенной ключевыми словами BEGIN и END.
- Предложение RETURNS содержит единственную инструкцию SELECT в круглых скобках. Результирующий набор инструкции SELECT формирует таблицу, возвращенную функцией. При использовании инструкции SELECT во встроенных функциях она подчиняется тем же ограничениям, что и при использовании в представлениях.
- Возвращающая табличное значение функция работает только с константами или аргументами **@local_variable**.

Пример:

```
CREATE FUNCTION Sales.ufn_SalesByStore (@storeid int)
RETURNS TABLE AS RETURN
( SELECT P.ProductID, P.Name, SUM(SD.LineTotal)
  AS 'YTD Total‘ FROM Production.Product AS P
  JOIN Sales.SalesOrderDetail AS SD ON
      SD.ProductID = P.ProductID
  JOIN Sales.SalesOrderHeader AS SH ON
      SH.SalesOrderID = SD.SalesOrderID
  WHERE SH.CustomerID = @storeid
  GROUP BY P.ProductID, P.Name );
```

Результат:

```
Product ID  Name                    YTD Total
----------------------------------------------
707         Sport-100 Helmet, Red     620.250910
708         Sport-100 Helmet, Black   657.636610
```

4.3. Объединение и ограничение результирующих наборов

Объединение с помощью UNION

Объединяет результаты двух или более запросов в один результирующий набор, в который входят все строки, принадлежащие всем запросам в объединении. Операция UNION отличается от соединений столбцов из двух таблиц.

Ниже приведены основные правила объединения результирующих наборов двух запросов с помощью операции UNION:

- Количество и порядок столбцов должны быть одинаковыми во всех запросах.
- Тип данных должен быть совестимым.

Пример:

```
SELECT * FROM testa
UNION ALL
SELECT * FROM testb;
```

Результат:

```
columna   columnb
------------------
100       test
100       test
...
(8 row(s) affected)
```

Ограничение результирующего набора с помощью EXCEPT и INTERSECT

Операторы EXCEPT и INTERSECT возвращают различные значения, сравнивая результаты двух запросов.

Оператор EXCEPT возвращает все различные значения, возвращенные левым запросом и отсутствующие в результатах выполнения правого запроса.

Оператор INTERSECT возвращает все различные значения, входящие в результаты выполнения, как левого, так и правого запроса.

Основные правила объединения результирующих наборов двух запросов с оператором EXCEPT или INTERSECT таковы:

•количество и порядок столбцов должны быть одинаковыми во всех запросах;

•типы данных должны быть совместимыми.

Если типы данных сравниваемых столбцов, возвращенных запросами, указанными слева и справа от оператора EXCEPT или INTERSECT, являются символьными типами с разными режимами сопоставления, сравнение выполняется в соответствии с правилами очередности параметров сортировки. Если нужное преобразование выполнить не удается, компонент SQL Server Database Engine возвращает ошибку.

Если сравниваются строки с целью определения различных значений, два значения NULL считаются равными.

Имена столбцов в результирующем наборе, возвращаемом оператором EXCEPT или INTERSECT, совпадают с именами, возвращаемыми запросом, который указан слева от оператора.

Имена столбцов или псевдонимы в предложениях ORDER BY должны ссылаться на имена столбцов, возвращаемых запросом, указанным слева от оператора.

Возможность хранения пустых значений в каком-либо столбце результирующего набора, возвращаемого оператором EXCEPT или INTERSECT, зависит от того, поддерживает ли это соответствующий столбец, возвращаемый запросом, указанным слева от оператора.

Пример EXCEPT:

```
SELECT ProductID
FROM Production.Product
EXCEPT
SELECT ProductID
FROM Production.WorkOrder
```

Результат:

```
ProductID
------------------
429
...
(266 row(s) affected)
```

Пример INTERSECT:

```
SELECT ProductID
FROM Production.Product
INTERSECT
SELECT ProductID
FROM Production.WorkOrder
```

Результат:

```
ProductID
-----------------
3
...
(238 row(s) affected)
```

Определение порядка очередности UNION, EXCEPT, INTERSECT

Если оператор EXCEPT или INTERSECT используется в выражении вместе с другими операторами, оно обрабатывается в следующем порядке:

1. Выражения в скобках.
2. Оператор INTERSECT.
3. Операторы EXCEPT и UNION обрабатываются слева направо в соответствии с их позицией в выражении.

Если оператор EXCEPT или INTERSECT используется для сравнения более двух наборов запросов, преобразование типов данных выполняется на основе сравнения двух запросов сразу с соблюдением вышеупомянутых правил обработки выражений.

Операторы EXCEPT и INTERSECT нельзя использовать в определениях распределенных секционированных представлений, в уведомлениях об изменениях результатов запроса и вместе с предложениями COMPUTE и COMPUTE BY.

Операторы EXCEPT и INTERSECT можно применять в распределенных запросах, но они будут выполнены только на локальном сервере и не будут распространены на связанный сервер. Таким образом, использование операторов EXCEPT и INTERSECT в распределенных запросах может сказаться на производительности.

При использовании в операции EXCEPT или INTERSECT в результирующем наборе полностью поддерживаются быстрые однонаправленные и статические курсоры. Если в операции EXCEPT или INTERSECT применяется курсор, управляемый набором ключей, или динамический курсор, то курсор результирующего набора преобразуется в статический курсор.

При выводе данных об операции EXCEPT с помощью средства графического отображения плана в среде SQL Server Management Studio операция представляется как left anti semi join, а операция INTERSECT — как left semi join.

Ограничение результирующего набора с использованием TOP и TABLESAMPLE

Чтобы ограничить количество строк, возвращаемых в результирующий набор, можно использовать предложение TOP.

Если в инструкции SELECT вместе с предложением TOP также содержится предложение ORDER BY, возвращаемые строки выбираются из упорядоченного результирующего набора. Весь результирующий набор строится в определенном порядке, и возвращаются верхние строки n этого набора. Если также указывается WITH TIES, возвращаются все строки, содержащие последнее значение, возвращенное предложением ORDER BY, даже если их количество превышает число, указанное expression.

Еще одним способом ограничения размеров результирующего набора является предварительное выполнение инструкции SET ROWCOUNT n. Есть ряд различий между предложениями SET ROWCOUNT и TOP:

- Предложение TOP применяется только к той инструкции SELECT, в которой оно указано. SET ROWCOUNT продолжает действовать до выполнения другой инструкции SET ROWCOUNT, например SET ROWCOUNT 0, которая выключит этот параметр.

Важно!

Использование инструкции SET ROWCOUNT не повлияет на работу инструкций DELETE, INSERT и UPDATE в следующем выпуске SQL Server. При программировании избегайте использования инструкции SET ROWCOUNT с инструкциями DELETE, INSERT и UPDATE и постарайтесь внести изменения в приложения, которые используют ее в настоящее время. Рекомендуется заменить инструкцию SET ROWCOUNT на предложение TOP для инструкций DELETE, INSERT и UPDATE.

- Несмотря на то, что действие SET ROWCOUNT на инструкцию SELECT остается прежним, предпочтительно с инструкцией SELECT использовать TOP по ряду причин:
- Использование инструкции SET ROWCOUNT приводит к тому, что большинство инструкций SELECT, INSERT, UPDATE и DELETE перестают выполняться, обработав указанное количество строк. Такое поведение применимо и по отношению к срабатыванию триггеров.
- Как часть инструкции SELECT, в процессе формирования плана выполнения запроса оптимизатор запросов может использовать значение expression в предложении TOP. Учитывая то, что SET ROWCOUNT используется вне инструкции, выполняющей запрос, его значение не может быть использовано для формирования плана запроса.

Пример TOP:

```
SELECT TOP (15)
    FirstName, LastName
FROM Person.Person
```

Результат:

```
FirstName  LastName
--------------------
Syed       Abbas
Catherine  Abel
...
(15 row(s) affected)
```

Предложение TABLESAMPLE ограничивает количество строк, возвращенных из таблицы в предложении FROM, указанным числом или процентом. Например:

TABLESAMPLE не может применяться к производным таблицам, таблицам на связанных серверах и таблицам, производным от возвращающих табличное значение функций, а также от функций, возвращающих наборы строк, и инструкций OPENXML. Предложение TABLESAMPLE нельзя указать в определении представления или во встроенной возвращающей табличное значение функции.

TABLESAMPLE может использоваться для быстрой выборки из большой таблицы, если выполняется одно из следующих условий.

- Выборка не является по-настоящему случайной на уровне отдельных строк.
- Строки на отдельных страницах таблицы не соотносятся с другими строками на одной и той же странице.

Пример TABLESAMPLE:

```
SELECT
    FirstName, LastName
FROM Person.Person
TABLESAMPLE (1 PERCENT)
```

Результат:

```
FirstName  LastName
--------------------
Eduardo    Barnes
Edward     Barnes
...
(199 row(s) affected)
```

5. Работа с подзапросами

5.1. Создание основных подзапросов

Подзапрос

Вложенным запросом называется запрос, помещаемый в инструкцию SELECT, INSERT, UPDATE или DELETE или в другой вложенный запрос. Подзапрос может быть использован везде, где разрешены выражения.

Вложенный запрос по-другому называют внутренним запросом или внутренней операцией выбора, в то время как инструкцию, содержащую вложенный запрос, называют внешним запросом или внешней операцией выбора.

Многие инструкции языка Transact-SQL, включающие подзапросы, можно записать в виде соединений. Другие запросы могут быть осуществлены только с помощью подзапросов. В языке Transact-SQL обычно не бывает разницы в производительности между инструкцией, включающей вложенный запрос, и семантически эквивалентной версией без вложенного запроса. Однако в некоторых случаях, когда проверяется существование, соединения показывают

лучшую производительность. В противном случае для устранения дубликатов вложенный запрос должен обрабатываться для получения каждого результата внешнего запроса. В таких случаях метод работы соединений дает лучшие результаты.

Запрос SELECT вложенного запроса всегда заключен в скобки. Он не может включать предложений COMPUTE или FOR BROWSE и может включать предложение ORDER BY только вместе с предложением TOP.

Вложенный запрос может быть вложен в предложение WHERE или HAVING внешней инструкции SELECT, INSERT, UPDATE или DELETE или в другой вложенный запрос. Возможно создавать вложенность до 32-го уровня, хотя ограничения меняются в зависимости от объема доступной памяти и сложности других выражений в запросе. Отдельные запросы могут не поддерживать вложенность до 32-го уровня. Подзапрос может появляться везде, где может использоваться выражение, если он возвращает одно значение.

Если таблица появляется только во вложенном запросе, а не во внешнем запросе, в этом случае столбцы данной таблицы не могут быть включены в выходные данные (список выборки внешнего запроса).

Инструкции, включающие вложенные запросы, обычно имеют один из следующих форматов:

- WHERE expression [NOT] IN **(**subquery**)**
- WHERE expression comparison_operator [ANY | ALL] **(**subquery**)**
- WHERE [NOT] EXISTS **(**subquery**)**

В некоторых инструкциях языка Transact-SQL вложенный запрос может рассматриваться как отдельный запрос. По существу, результаты вложенного запроса подставляются во внешний запрос (хотя это необязательно и зависит от того, как в MicrosoftSQL Server реализована обработка инструкций языка Transact-SQL с вложенными запросами).

Существуют три основных типа подзапросов, которые:

- работают в списках, вставленных после ключевого слова IN, или тех, которые оператор сравнения изменил с помощью ключевого слова ANY или ALL;
- вставлены оператором немодифицированных сравнений и должны возвращать одно значение;
- являются тестами на существование, начинающимися с ключевого слова EXISTS.

Пример:

```
SELECT ProductID, Name
FROM Production.Product
WHERE Color NOT IN
    (SELECT Color
    FROM Production.Product
    WHERE ProductID = 5)
```

Результат:

```
ProductID    Name
-----------------------------------
1                    Adjustable Race
2                    Bearing Ball
...
(504 row(s) affected)
```

Подзапрос в качестве выражения

В языке Transact-SQL вложенный запрос может быть заменен в любом месте, где выражение может использоваться в инструкции SELECT, UPDATE, INSERT и DELETE, за исключением списка ORDER BY.

Пример:

```
SELECT Name, ListPrice,
(SELECT AVG(ListPrice) FROM Production.Product)
      AS Average, ListPrice –
(SELECT AVG(ListPrice) FROM Production.Product)
   AS Difference
FROM Production.Product
WHERE ProductSubcategoryID = 1
```

Результат:

```
Name                        ListPrice  Average   Difference
-----------------------------------------------------
Mountain-100 Silver, 38  3399.99    438.6662  2961.3238
Mountain-100 Silver, 42  3399.99    438.6662  2961.3238
...
(32 row(s) affected)
```

Использование операторов ANY, ALL, и SOME

Операторы сравнения с вложенными запросами могут быть уточнены с помощью ключевых слов ALL или ANY. SOME является эквивалентом ANY в стандарте ISO.

Вложенные запросы операторов сравнения с модификаторами возвращают список из нуля или более значений и могут включать предложения GROUP BY или HAVING. Эти вложенные запросы могут быть переформулированы с использованием ключевого слова EXISTS.

Пример ANY:

```
SELECT Name
FROM Production.Product
WHERE ListPrice >= ANY
    (SELECT MAX (ListPrice)
     FROM Production.Product
     GROUP BY
ProductSubcategoryID)
```

Результат:

```
Name
--------------------
LL Mountain Seat
ML Mountain Seat ...
(304 row(s) affected)
```

Пример ALL:

```
SELECT Name
FROM Production.Product
WHERE ListPrice >= ALL
    (SELECT MAX (ListPrice)
     FROM Production.Product
     GROUP BY
ProductSubcategoryID)
```

Результат:

```
Name
--------------------
Road-150 Red, 62
Road-150 Red, 44...
(5 row(s) affected)
```

Сравнение скалярных и табличных подзапросов

Скалярный подзапроса подзапрос, который возвращает одну строку данных, в то время как табличный подзапрос возвращает несколько строк данных.

Пример скалярного подзапроса:

```
create table T1 (a int, b int)
create table T2 (a int, b int)
select *
from T1
where T1.a > (select max(T2.a)
from T2 where T2.b < T1.b)
```

Результат:

```
a           b
----------------------
...
(0 row(s) affected)
```

Пример табличного подзапроса:

```
SELECT Name
FROM Production.Product
WHERE ListPrice =
    (SELECT ListPrice
     FROM Production.Product
     WHERE Name = 'Chainring Bolts' )
```

Результат:

```
Name
------------------
Adjustable Race
Bearing Ball ...
(200 row(s) affected)
```

Правила создания подзапросов

На вложенный запрос распространяются следующие ограничения:

- Список выбора вложенного запроса, начинающийся с оператора сравнения, может включать только одно выражение или имя столбца (за исключением операторов EXISTS и IN, работающих в инструкции SELECT * или в списке соответственно).
- Если предложение WHERE внешнего запроса включает имя столбца, оно должно быть совместимо для соединения со столбцом в списке выбора вложенного запроса.
- Типы данных ntext, text и image не могут быть использованы в списке выбора вложенных запросов.
- Вложенные запросы, представленные оператором немодифицированного сравнения (после которого нет ключевого слова ANY или ALL), не могут включать предложения типа GROUP BY и HAVING, поскольку они должны возвращать одиночное значение.
- Ключевое слово DISTINCT не может быть использовано во вложенном запросе, включающем предложение GROUP BY.
- Нельзя указывать предложения COMPUTE и INTO.
- Предложение ORDER BY может быть указано только вместе с предложением TOP.
- Представление, созданное с помощью вложенного запроса, не может быть обновлено.
- Список выбора вложенного запроса, начинающегося с предложения EXISTS, по соглашению содержит звездочку (*) вместо отдельного имени столбца. Правила для вложенного запроса, начинающегося с предложения EXISTS, являются такими же, как для стандартного списка выбора, поскольку вложенный запрос, начинающийся с предложения EXISTS, проводит проверку существования и возвращает TRUE или FALSE вместо данных.

Уточнение имен столбцов во вложенных запросах:

Общее правило состоит в том, что имена столбцов в инструкции неявно уточняются именем таблицы, указанной в предложении FROM того же уровня вложенности. Если столбец не существует в таблице, на которую ссылается предложение FROM вложенного запроса, он неявно уточняется именем таблицы, указанной во внешнем запросе.

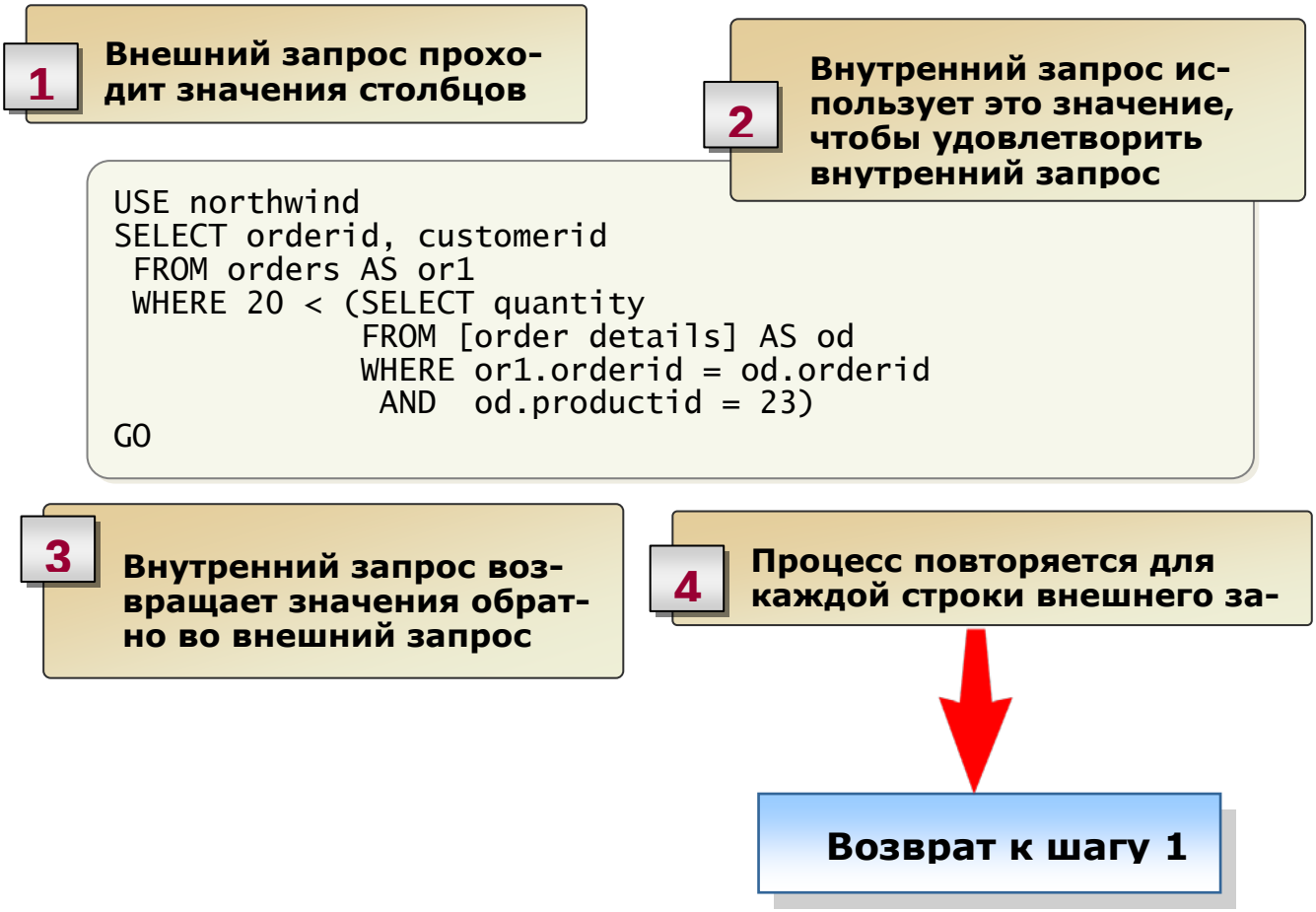
5.2.Связанные подзапросы

Связанные подзапросы

Результат многих запросов может быть получен путем выполнения одного вложенного запроса и подстановки полученного результата или результатов в предложение WHERE внешнего запроса. В запросах, содержащих коррелированные вложенные запросы (также называемые повторяющимися вложенными

запросами), вложенный запрос зависит по значению от внешнего запроса. Это означает, что выполнение вложенного запроса повторяется по одному разу для каждой строки, которая может быть выбрана внешним запросом.

Пример:



Разработка связанных подзапросов

Пример:

Внешний запрос

```
SELECT c.LastName,
c.FirstName
FROM Person.Person c
JOIN HumanResources.Employee
e
 ON e.BusinessEntityID =
     c.BusinessEntityID
```

Внутренний запрос

```
SELECT Bonus
FROM Sales.SalesPerson
```

Связанный запрос

```
SELECT c.LastName, c.FirstName
FROM Person.Person c JOIN HumanResources.Employee
e
ON e.BusinessEntityID = c.BusinessEntityID
WHERE 5000.00 IN
(SELECT Bonus
FROM Sales.SalesPerson sp
WHERE e.BusinessEntityID = sp.BusinessEntityID) ;
```

В примере имеем внешний запрос, который извлекает имя и фамилию каждого сотрудника.

Внутренний запрос, получает каждый бонус для всех продавцов. Добавление его внешний запрос создаст коррелированный подзапрос, который появляется в следующем шаге. Обратите внимание, что это внутренние запросы не должны переписываться для того, чтобы менять смысл в коррелированных подзапросов.

В связанном подзапросе объединили два запроса в коррелированном подзапросе. Этот запрос получает один экземпляр имя и фамилию каждого сотрудника, для которого бонус в SalesPerson таблицы 5000 и идентификационный номер(BusinessEntityID) совпадает с сотрудником в SalesPerson. SQL Server считает каждую строку таблицы сотрудников для включения в результаты, подставляя значения в каждой строке во внутреннем запросе.

Использование связанных подзапросов

Коррелированные подзапросы также известны как повторяющиеся подзапросы, где подзапрос зависит от внешнего запроса. Это означает, что подзапрос

выполняется неоднократно, один раз для каждой строки, которая может быть выбрана из внешнего запроса.

Этот запрос получает один экземпляр имя и фамилию каждого сотрудника, для которого бонус в SalesPerson таблице 5000,00 и для которых ИД сотрудника совпадает с ИД сотрудника в таблице SalesPerson.

Пример:

```
SELECT DISTINCT c.LastName, c.FirstName
FROM Person.Person c JOIN HumanResources.Employee e
ON e.BusinessEntityID = c.BusinessEntityID
WHERE 5000.00 IN
(SELECT Bonus
FROM Sales.SalesPerson sp
WHERE e.BusinessEntityID = sp.BusinessEntityID);
```

Результат:

```
LastName                FirstName
--------------------------
Ansman-Wolfe            Pamela
Saraive                      José
(2 row(s) affected)
```

Использование EXISTS со связанными подзапросами

Если вложенный запрос вводится с помощью ключевого слова EXISTS, то он выступает в роли проверки на существование. В предложении WHERE внешнего запроса проверяется факт существования строк, возвращенных вложенным запросом. Вложенный запрос не выдает никаких данных, а возвращает значение TRUE или FALSE.

Обратите внимание на то, что вложенные запросы, введенные с помощью ключевого слова EXISTS, отличаются от других вложенных запросов следующим образом.

- Перед ключевым словом EXISTS не ставится имя столбца, константы или другого выражения.
- Список выборки для вложенного запроса, введенного с помощью ключевого слова EXISTS, практически всегда состоит из знаков «звездочка» (*). Так как производится проверка только на предмет существования строк, удовлетворяющих заданным во вложенном запросе условиям, то нет необходимости выводить имена найденных столбцов.

Ключевое слово EXISTS часто является единственным способом провести необходимую проверку. Хотя некоторые запросы, созданные с помощью

ключевого слова EXISTS, не могут быть выражены по-другому, многие запросы для достижения подобных результатов используют IN или оператор сравнения, измененный с помощью ANY либо ALL.

Пример:

```
SELECT Name
FROM Production.Product
WHERE EXISTS
   (SELECT * FROM Production.ProductSubcategory
    WHERE ProductSubcategoryID = Production.Product.ProductSubcategoryID
    AND Name = 'Wheels')
```

Результат:

```
Name
----------------------
LL Mountain Front Wheel
ML Mountain Front Wheel
...
(14 row(s) affected)
```

5.3. Сравнение подзапросов с объединениями и временными таблицами

Сравнение подзапросов с объединениями

С помощью соединения можно получать данные из двух или нескольких таблиц на основе логических связей между ними. Соединения указывают, как MicrosoftSQL Server должен использовать данные из одной таблицы для выбора строк из другой таблицы.

Соединение определяет способ связывания двух таблиц в запросе следующим образом:

- для каждой таблицы указываются столбцы, используемые в соединении. В типичном условии соединения указывается внешний ключ из одной таблицы и связанный с ним ключ из другой таблицы;
- указывается логический оператор (например, = или <>,) для сравнения значений столбцов.

Внутреннее соединение можно задавать в предложениях FROM и WHERE. Внешнее соединение можно указывать только в предложении FROM. Условия соединения сочетаются с условиями поиска WHERE и HAVING для управления строками, выбранными из базовых таблиц, на которые ссылается предложение FROM.

То, что условия соединения задаются в предложении FROM, помогает отделить их от условий поиска, которые могут быть заданы в предложении

WHERE. Объединение рекомендуется задавать именно таким способом. Ниже приведен упрощенный синтаксис соединения с использованием предложения FROM стандарта ISO:

FROM first_tablejoin_typesecond_table [ON (join_condition)]

Аргумент join_type задает тип соединения: внутренний, внешний или перекрестный. Аргумент join_condition определяет предикат, который будет вычисляться для каждой пары соединяемых строк.

Вложенным запросом называется запрос, помещаемый в инструкцию SELECT, INSERT, UPDATE или DELETE или в другой вложенный запрос. Подзапрос может быть использован везде, где разрешены выражения. В данном примере вложенный запрос используется в качестве выражения для столбца с именем MaxUnitPrice в инструкции SELECT.

Вложенный запрос по-другому называют внутренним запросом или внутренней операцией выбора, в то время как инструкцию, содержащую вложенный запрос, называют внешним запросом или внешней операцией выбора.

Многие инструкции языка Transact-SQL, включающие подзапросы, можно записать в виде соединений. Другие запросы могут быть осуществлены только с помощью подзапросов. В языке Transact-SQL обычно не бывает разницы в производительности между инструкцией, включающей вложенный запрос, и семантически эквивалентной версией без вложенного запроса. Однако в некоторых случаях, когда проверяется существование, соединения показывают лучшую производительность. В противном случае для устранения дубликатов вложенный запрос должен обрабатываться для получения каждого результата внешнего запроса. В таких случаях метод работы соединений дает лучшие результаты

Временные таблицы

Кроме стандартной роли обычных определяемых пользователем таблиц SQL Server предоставляет в базах данных следующие типы таблиц специального назначения:

- Секционированные таблицы
- Временные таблицы
- Системные таблицы
- Широкие таблицы

Существует два вида временных таблиц: локальные и глобальные. Локальные временные таблицы видны только их создателям до завершения сеанса соединения с экземпляром SQL Server, как только они впервые созданы или когда на них появляется ссылка.

Локальные временные таблицы удаляются после отключения пользователя от экземпляра SQL Server.

Пример:

```
CREATE TABLE #StoreInfo  (
  EmployeeID int,
  ManagerID int,
  Num int  )
```

Глобальные временные таблицы видны всем пользователям в течение любых сеансов соединения после создания этих таблиц и удаляются, когда все пользователи, ссылающиеся на эти таблицы, отключаются от экземпляра SQL Server.

Пример:

```
CREATE TABLE ##StoreInfo (
  EmployeeID int,
  ManagerID int,
  Num int )
```

Временные таблицы отличаются от постоянных только тем, что хранятся в базе данных **tempdb** и автоматически удаляются, когда необходимость в них отпадает.

Локальные и глобальные временные таблицы отличаются друг от друга именами, видимостью и доступностью. Имена локальных временных таблиц начинаются с одного символа (#); они видны только текущему соединению пользователя и удаляются, когда пользователь отключается от экземпляра SQL Server. Имена глобальных таблиц начинаются с двух символов номера (##); они видны любому пользователю и удаляются, когда все пользователи, которые на них ссылаются, отключаются от экземпляра SQL Server.

Например, если создать таблицу **employees**, она будет доступна любому пользователю, которому предоставлены разрешения на ее использование до тех пор, пока не будет удалена. Если во время сеанса базы данных создается локальная временная таблица **#employees**, с ней сможет работать только данный сеанс. Таблица будет удалена при завершении сеанса. Если создать глобальную временную таблицу **##employees**, с ней сможет работать любой пользователь базы данных. Если другие пользователи не будут работать с этой таблицей, она будет удалена после отключения от нее. Если другой пользователь обратится к созданной таблице, SQL Server удалит ее, когда произойдет отключение и другие сеансы перестанут активно к ней обращаться.

В большинстве случаев временные таблицы можно заменить переменными типа **table**.

Сравнение подзапросов с временными таблицами

При проектировании базы данных необходимо добиться, чтобы все важные операции выполнялись правильно и быстро. Некоторые проблемы произ-

водительности можно решить после того, как база данных будет помещена в производственную среду. Тем не менее, причиной многих проблем могут стать ошибки при проектировании базы данных, которые можно исправить только путем изменения ее структуры.

При проектировании и реализации базы данных следует выделить большие таблицы и наиболее сложные процессы. При их разработке необходимо уделить особое внимание производительности. Кроме того, следует определить, какое влияние на производительность будет оказывать увеличивающееся количество пользователей, обращающихся к таблицам.

Ниже продемонстрировано несколько примеров изменения структуры базы данных для улучшения производительности.

- Если требуется ежедневно вычислять сводные данные по таблице, содержащей сотни тысяч строк, в нее можно добавить столбец или столбцы для хранения предыдущих данных, подвергшихся статистической обработке, и использовать их только при подготовке отчета.
- Базы данных могут быть излишне нормализированы. Это означает, что база данных содержит несколько небольших взаимосвязанных таблиц. При обработке данных в этих таблицах требуются дополнительные действия по объединению взаимосвязанных данных. Дополнительная обработка уменьшает производительность базы данных. В этом случае за счет денормализации базы данных можно упростить обработку и слегка увеличить производительность.

5.4. Использование общих табличных выражений

Табличное выражение

Обобщенные табличные выражения (ОТВ) можно представить себе как временные результирующие наборы, определенные в области выполнения единичных инструкций SELECT, INSERT, UPDATE, DELETE или CREATE VIEW. ОТВ, как и производные таблицы, не сохраняются в базе данных в виде объектов, время их жизни ограничено продолжительностью запроса. Но, в отличие от производных таблиц, ОТВ могут ссылаться сами на себя, а на них один и тот же запрос может ссылаться несколько раз.

ОТВ предназначены для:

- Создания рекурсивных запросов.
- Замены представлений в тех случаях, когда использование представления не оправдано, то есть тогда, когда нет необходимости сохранять в метаданных базы его определение.
- Группирования по столбцу, производного от скалярного подзапроса выборки или функции, которая недетерминирована или имеет внешний доступ.
- Многократных ссылок на результирующую таблицу из одной и той же инструкции.

Применение ОТВ позволяет значительно повысить читаемость и упростить работу со сложными запросами, разбив его на отдельные логические

строительные блоки. Из них можно составлять более сложные промежуточные ОТВ для формирования конечного результирующего набора.

ОТВ могут быть определены в пользовательских подпрограммах (функциях, хранимых процедурах, триггерах, представлениях).

ОТВ состоит из имени выражения, необязательного списка столбцов и определяющего ОТВ запроса. После определения ОТВ на него можно ссылаться из инструкций SELECT, INSERT, UPDATE и DELETE как на таблицу или представление. ОТВ также можно указать в предложении CREATE VIEW в определяющей инструкции SELECT.

Пример:

```
WITH TopSales (SalesPersonID, NumSales) AS
( SELECT SalesPersonID, Count(*)
  FROM Sales.SalesOrderHeader GROUP BY SalesPersonId )
SELECT * FROM TopSales
WHERE SalesPersonID IS NOT NULL
ORDER BY NumSales DESC
```

Создание временных табличных выражений

Использование ОТВ предлагает преимущества улучшения читаемости и простоты в обслуживании сложных запросов. Запрос может быть разделен на отдельные, простые, логические блоки. Эти простые блоки могут быть использованы для построения более сложных, промежуточных ОТВ, пока на наберется окончательный результат.

Чтобы построить ОТВ:

1. Выберите имя для ОТВ и список столбцов
2. Создайте ОТВ SELECT запрос
3. Используйте ОТВ в запросе

Пример:

```
WITH TopSales (SalesPersonID, NumSales) AS
(SELECT SalesPersonID, Count(*)
 FROM Sales.SalesOrderHeader GROUP BY SalesPersonId)
SELECT LoginID, NumSales
FROM HumanResources.Employee e INNER JOIN TopSales
ON TopSales.SalesPersonID = e.EmployeeID
ORDER BY NumSales DESC
```

Рекурсивные запросы с использованием общих табличных выражений

Табличное выражения (ОТВ) обеспечивает значительное преимущество, которое позволяет ссылаться на себя, тем самым создавая рекурсивное ОТВ.

Рекурсивные ОТВ, это ОТВ, которые неоднократно выполняться, чтобы вернуться подмножества данных до полного набора результата.

Для создания рекурсивного ОТВ:

1. Создайте запрос корня дерева (вершины рекурсивного дерева)
2. Добавьте оператор UNION ALL
3. Создайте запрос рекурсивного обхода, который использует ссылки на CTE

Пример:

```
SELECT ManagerID, EmployeeID
FROM HumanResources.Employee
WHERE ManagerID IS NULL
UNION ALL
SELECT e.ManagerID, e.EmployeeID
FROM HumanResources.Employee e
INNER JOIN HumanResources.Employee mgr
ON e.ManagerID = mgr.EmployeeID
```

6. Модификация данных в таблицах

6.1. Вставка данных в таблицы

Основные сведения об инструкции INSERT

Добавлять строки в таблицу с помощью инструкций INSERT и SELECT можно следующими способами:

- При помощи инструкции INSERT, задав значения непосредственно или из вложенного запроса
- При помощи инструкции SELECT с предложением INTO.

Инструкция INSERT добавляет в таблицу одну или несколько новых строк. В простейшем виде инструкция INSERT имеет следующий вид:

INSERT [INTO] table_or_view [(column_list)] data_values

Инструкция INSERT выполняет вставку в указанную таблицу или представление значений data_values в виде одной или нескольких строк. Параметр column_list представляет собой разделяемый запятыми список имен столбцов, для которых представляются данные. Если аргумент column_list не задается, данные получают все столбцы таблицы или представления.

Если в аргументе column_list указаны не все столбцы таблицы или представления, то пропущенные столбцы вставляется либо значение по умолчанию (если для столбца оно задано), или значение NULL. Для всех пропущенных столбцов должно быть либо определено значение по умолчанию, либо допускаться значение NULL.

Значения для следующих типов столбцов формирует компонент SQL Server Database Engine, поэтому они в инструкции INSERT не указываются:

- Столбцы со свойством IDENTITY, формирующим значения для столбца.
- Столбцы, имеющие значения по умолчанию, используемые функцией NEWID для формирования уникального значения идентификатора GUID.
- Вычисляемые столбцы.

Пример:

Этот пример вставляет одну строку в таблицу Production.UnitMeasure. Так как значения для всех столбцов предоставлены и перечислены в том же порядке, что и столбцы таблицы, имена столбцов не указаны в column_list.

```
INSERT INTO Production.UnitMeasure
VALUES (N'F2', N'Square Feet', GETDATE());
```

Пример:

Вставка нескольких строк данных. Этот пример вставляет несколько строк в таблицу Production.UnitMeasure. Значения ключевых слов определяют значения для одной строки таблицы. Значения задаются через запятую в виде списка скалярных выражений, тип данных, точность и масштаб должен быть таким же или неявно преобразовываться в соответствующий столбец в списке столбцов. Если список столбцов не задан, то значения должны быть указаны в той же последовательности, что и столбцы в таблице или представлении.

```
INSERT INTO Production.UnitMeasure
VALUES (N'F2', N'Square Feet', GETDATE()), (N'Y2', N'Square Yards',
GETDATE());
```

Использование INSERT

Выражение INSERT с использованием SELECT может быть использовано для добавления значений в таблицу из одной или нескольких таблиц или представлений. Использование подзапроса SELECT, также позволяет вставлять более чем одну строку. Список выборки подзапроса должен соответствовать списку столбцов в выражении INSERT. Если список столбцов не задан, список выбора должен соответствовать столбцам таблицы или представления в которые вставляются данные.

Пример: INSERT с использованием SELECT

```
INSERT INTO MyTable (PriKey, Description)
    SELECT ForeignKey, Description
    FROM SomeView
```

Выражение INSERT использованием EXECUTE используется для получения некоторого набора данных с помощью хранимой процедуры, а затем сохранения результата в новой таблице.

Пример: INSERT с использованием EXECUTE

```
CREATE PROCEDURE dbo.SomeProcedure
INSERT dbo.SomeTable
EXECUTE SomeProcedure
```

Вставка с использованием выражения TOP используется для добавления верхних N строк одной таблицы другой.

Пример: INSERT с использованием TOP

```
INSERT TOP (#) INTO SomeTableA
    SELECT SomeColumnX, SomeColumnY
    FROM SomeTableB
```

Вставка значений в идентифицирующий столбец

При значении IDENTITY столбца, Microsoft SQL Server автоматически создает последовательные номера для новых строк, вставляемых в таблицу, содержащую столбец идентификаторов. Выражение INSERT вообще не указывается значения для столбцов с типом IDENTITY, потому что значения для этих столбцов создает SQL Server Database Engine. Однако, если значение идентификатора почему-то стали дублироваться или столбец идентификаторов поврежден, можно переопределить значения идентификаторов с помощью выражения SET IDENTITY_INSERT.

Пример показывает один из способов вставки данных в столбце идентификаторов. Первые два оператора INSERT не указывают значения идентификаторов, которые будут созданы для новой строки. Третий оператор INSERT переопределяет свойство IDENTITY для столбца с выражением SET IDENTITY_INSERT и вставляет явное значение в столбце идентификаторов.

Пример:

```
CREATE TABLE dbo.T1 ( column_1 int IDENTITY, column_2 VARCHAR(30));
GO
INSERT T1 VALUES ('Row #1');
INSERT T1 (column_2) VALUES ('Row #2');
GO
SET IDENTITY_INSERT T1 ON;
```

```
GO
INSERT INTO T1 (column_1,column_2)
   VALUES (-99, 'Explicit identity value');
GO
SELECT column_1, column_2
FROM T1;
```

INSERT и OUTPUT

Чтобы вернуть вставленные строки как часть операции вставки можно использовать OUTPUT с выражением INSERT.

Результаты могут быть возвращены в обработке приложений для использования подтверждения вставки, архивирования и других подобных действий.

Пример:

```
INSERT SomeTable
OUTPUT dml_select_list INTO
(@table_variable | output_table) (column_list)
```

Пример запроса:

```
DECLARE @MyTableVar table( ScrapReasonID smallint,
                    Name varchar(50),
                    ModifiedDate datetime);
INSERT Production.ScrapReason
   OUTPUT INSERTED.ScrapReasonID, INSERTED.Name, INSERT-
ED.ModifiedDate
      INTO @MyTableVar
VALUES (N'Operator error', GETDATE());
```

6.2.Удаление данных из таблиц

Основы использования DELETE

Инструкция DELETE удаляет одну или несколько строк из таблицы или представления.

Упрощенный синтаксис инструкции DELETE имеет следующий вид.

DELETE table_or_view

FROM table_sources

WHERE search_condition

Аргумент table_or_view перечисляет таблицы или представления, из которых нужно удалить строки. Удаляются все строки из таблицы или представления table_or_view, которые соответствуют заданным в предложении WHERE условиям поиска. Если предложение WHERE не указано, удаляются все строки

в таблице или представлении table_or_view. Предложение FROM задает дополнительные таблицы или представления и условия соединения, которые могут быть использованы предикатами в условиях поиска предложения WHERE для определения строк, подлежащих удалению из таблицы или представления table_or_view. Строки не удаляются из таблиц, перечисленных в предложении FROM, а только из таблицы, указанной в table_or_view.

Таблица, из которой удалены все строки, остается в базе данных. Инструкция DELETE удаляет только строки из таблицы; таблица должна быть удалена из базы данных с помощью инструкции DROP TABLE.

Пример:

```
DELETE table_or_view
FROM table_sources
WHERE search_condition
```

Особенности использования DELETE

DELETE без использования WHERE

DELETE может быть использован без WHERE. При этом удаляются все строки таблицы без ограничений. В примере удаляются все строки из таблицы Sales.SalesPerson потому что условие WHERE определены.

Пример:

```
DELETE FROM Sales.SalesPerson;
```

DELETE с использованием подзапроса

DELETE может быть определен как подзапрос для того, чтобы удалить строки из базовой таблицы в зависимости от данных, хранящихся в другой таблице. В примере удаляются строки из таблицы SalesPersonQuotaHistory на основе года до даты продажи, указанных в таблице SalesPerson.

Пример:

```
DELETE FROM Sales.SalesPersonQuotaHistory
WHERE SalesPersonID IN
   (SELECT SalesPersonID
    FROM Sales.SalesPerson
    WHERE SalesYTD > 2500000.00);
```

DELETE с использованием TOP

DELETE может быть изменен с помощью TOP так же, как INSERT , для того, чтобы удалить некоторое количество или процент строк из таблицы. В примере удаляется 2,5 процента строк (27 строк) из таблицы ProductionInventory.

Пример:

```
DELETE TOP (2.5) PERCENT
FROM Production.ProductInventory;
```

Определение и использование TRANCATE

TRUNCATE удаляет все строки в таблице, не записывая в журнал удаление отдельных строк. Инструкция TRUNCATE TABLE похожа на инструкцию DELETE без предложения WHERE, однако TRUNCATE TABLE выполняется быстрее и требует меньших ресурсов системы и журналов транзакций.

Пример:

```
TRUNCATE TABLE HumanResources.JobCandidate
```

Сравнение TRUNCATE и DELETE

По сравнению с инструкцией DELETE, инструкция TRUNCATE TABLE обладает следующими преимуществами:

- Используется меньший объем журнала транзакций.

Инструкция DELETE производит удаление по одной строке, и заносит в журнал транзакций запись для каждой удаляемой строки. Инструкция TRUNCATE TABLE удаляет данные, освобождая страницы данных, используемые для хранения данных таблиц, и в журнал транзакций записывает только данные об освобождении страниц.

- Обычно используется меньшее количество блокировок.

Если инструкция DELETE выполняется с блокировкой строк, для удаления блокируется каждая строка таблицы. Инструкция TRUNCATE TABLE всегда блокирует таблицу и страницу, но не каждую строку.

- В таблице остается нулевое количество страниц, без исключений.

После выполнения инструкции DELETE в таблице могут все еще оставаться пустые страницы. Например, чтобы освободить пустые страницы в куче, необходима, как минимум, монопольная блокировка таблицы (LCK_M_X). Если операция удаления не использует блокировку таблицы, таблица (куча) будет содержать множество пустых страниц. Для индексов после операции удаления могут остаться пустые страницы, хотя эти страницы и будут быстро освобождены в процессе фоновой очистки.

Инструкция TRUNCATE TABLE удаляет все строки таблицы, но структура таблицы и ее столбцы, ограничения, индексы и т. п. сохраняются. Чтобы удалить не только данные таблицы, но и ее определение, следует использовать инструкцию DROP TABLE.

Если таблица содержит столбец идентификаторов, счетчик этого столбца сбрасывается до начального значения, определенного для этого столбца. Если начальное значение не задано, используется значение по умолчанию, равное 1. Чтобы сохранить столбец идентификаторов, используйте инструкцию DELETE.

Пример:

```
DELETE FROM Sales.SalesPerson;
```

```
TRUNCATE TABLE Sales.SalesPerson;
```

DELETE с OUTPUT

Возвращает данные из строк, изменившихся в результате выполнения инструкций INSERT, UPDATE, DELETE или MERGE, или выражения на основе этих данных. Результаты могут быть возвращены приложению, например, для вывода подтверждающих сообщений, архивирования и т. п. Результаты также могут быть вставлены в таблицу или табличную переменную. Кроме того, можно записать результаты предложения OUTPUT во вложенных инструкциях INSERT, UPDATE, DELETE или MERGE и вставить эти результаты в целевую таблицу или представление.

Инструкции UPDATE, INSERT или DELETE с предложением OUTPUT возвращают строки клиенту даже в случае, если при выполнении инструкции возникли ошибки и был выполнен ее откат. Результат не может быть использован, если при выполнении инструкции возникли какие-либо ошибки.

Пример:

```
DELETE Production.Culture
    OUTPUT DELETED.*;
```

Результат:

```
CultureID     Name        ModifiedDate
--------------------------------------
Ar            Arabic      1998-06-01
En            English     1998-06-01
```

6.3.Обновление данных в таблице

Основные сведения об UPDATE

Используя инструкцию UPDATE, можно изменять значения данных в отдельных строках, группах строк или во всех строках таблицы или представления. Инструкция может также применяться для обновления строк в удаленном сервере с использованием имени связанного сервера или функций OPENROWSET, OPENDATASOURCE и OPENQUERY, если поставщик OLE DB, используемый для доступа к удаленному серверу, поддерживает обновления. Инструкция UPDATE, которая ссылается на таблицу или представление, может одновременно изменять данные только в одной базовой таблице.

Инструкция UPDATE имеет следующие основные предложения:

- SET

Содержит разделенный запятыми список столбцов, подлежащих обновлению, и новые значения для каждого столбца, в форме column_name = expression. Значение, предоставляемое выражением, включает такие элементы, как константы, значения, выбранные из столбца в другой таблице или представлении или значения, вычисленные с использованием сложного выражения.

FROM

Выявляет таблицы или представления, которые поставляют значения для выражений в предложении SET, и необязательные условия соединения между исходными таблицами и представлениями.

WHERE

Указывает условие поиска, которое определяет строки исходных таблиц и представлений, пригодные для предоставления значений выражениям в предложении SET.

Пример UPDATE:

```
UPDATE Sales.SalesPerson
SET Bonus = 6000;
```

```
UPDATE Sales.SalesPerson
SET Bonus = Bonus * 2;
```

Пример UPDATE с WHERE

```
UPDATE Production.Product
SET Color = N’Metallic Red’
WHERE Name LIKE N’Road-250%’          AND Color = N’Red’;
```

Обновление данными из других таблиц

Для обновления строк в одной таблице можно выбрать данные из другой таблицы с помощью подзапроса.

Можно использовать подзапрос в предложении FROM UPDATE в место явного указания исходной таблицы, которая используется в качестве источника для операции обновления.

В примере изменяется столбец SalesYTD в таблице SalesPerson, чтобы отразить самые последние продажи, записанные в таблице SalesOrderHeader.

Обновление информации из другой таблицы могут быть полезны, когда есть несколько таблиц, которые связаны друг с другом с точки зрения бизнеса, такие, как например, продавцы в Adventure Works (которые находятся таблице SalesPerson). Продажи отражаются в таблице SalesOrderHeader, но непосредственно не влияют на столбец SalesYTD (год даты продажи) в таблице SalesPerson.

Данные ДО:

```
SalesYTD
-------------
677558.4653
4557045.0459
```

Пример:

```
UPDATE Sales.SalesPerson
SET SalesYTD = SalesYTD + SubTotal
FROM Sales.SalesPerson AS sp
JOIN Sales.SalesOrderHeader AS so
   ON sp.BusinessEntityID = so.SalesPersonID
   AND so.OrderDate = (SELECT MAX(OrderDate)
                FROM Sales.SalesOrderHeader
                WHERE SalesPersonID =
                    sp.BusinessEntityID);
```

Результат ПОСЛЕ:

```
SalesYTD
-------------
721382.488
4593234.5123
```

UPDATE с использованием OUTPUT

Чтобы вернуть обновленные строки как часть операции обновления можно использовать OUTPUT с UPDATE. Результаты могут быть возвращены в панель результатов SQL Server Management Studio для просмотра или в таблич-

ную переменную для использования в при подтверждении сообщения, архивирования и других подобных операций.

В примере возвращается обновленное значение столбца в таблице INSERTED.Bonus в переменную @ NewTableVar. Затем используется SELECT для возврата значения в @ NewTableVar чтобы убедиться, что значения в таблице SalesPerson данные на самом деле были обновлены.

Использование выходных данных с UPDATE может быть полезно для возвращения результатов из одного запроса с UPDATE в табличную переменную, которая затем может быть использована в других запросах или в приложениях, которые могут получить доступ к данным SQL Server.

Пример:

```
DECLARE @NewTableVar table ( Dollars money );
UPDATE Sales.SalesPerson
SET Bonus = 10000
OUTPUT INSERTED.Bonus INTO @NewTableVar;
SELECT Dollars
FROM @NewTableVar;
```

Результат:

```
Dollars
--------
10000.00
10000.00
...
(17 row(s) affected)
```

6.4. Обзор транзакций

Основные сведения о транзакциях

Транзакция является последовательностью операций, выполненных как одна логическая единица работы. Логическая единица работы должна обладать четырьмя свойствами, называемыми атомарностью, согласованностью, изоляцией и длительностью (ACID), чтобы называться транзакцией.

Атомарность

Транзакция должна быть атомарной единицей работы; должны быть выполнены либо все входящие в нее модификации данных, либо ни одно из них не должно быть выполнено.

Согласованность

По завершении, транзакция должна оставить все данные в согласованном состоянии. В реляционной базе данных к модификациям транзакции должны быть применены все правила для обеспечения целостности всех данных. Все

внутренние структуры данных, например индексы сбалансированного дерева или взаимосвязанные списки, должны быть правильными в конце транзакции.

Изоляция

Модификации, выполняемые параллельной транзакцией, должны быть изолированы от любых модификаций, проводимых другими параллельными транзакциями. Транзакция распознает данные либо в состоянии, в котором они были до того, как другая параллельная транзакция изменила их, либо она распознает данные после того, как другая транзакция была завершена. Но она не распознает промежуточное состояние. Это упоминается как упорядоченность, потому что она обеспечивает возможность перезагрузить начальные данные и воспроизвести серию транзакций, чтобы завершить работу с данными в том же самом состоянии, в котором они были после выполнения исходных транзакций.

Длительность

После завершения транзакции, произведенные ею действия занимают постоянное место в системе. Изменения сохраняются даже в случае системного сбоя.

Транзакции и Database Engine

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

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

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

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

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

Основные инструкции транзакций

Инструкция BEGIN TRANSACTION отмечает начальную точку явной локальной транзакции, увеличивает значение функции @@TRANCOUNT на 1.

Инструкция BEGIN TRANSACTION предоставляет точку, где гарантируется логическая и физическая согласованность данных, на которые ссылается соединение. В случае ошибок для всех изменений после BEGIN TRANSACTION можно выполнить откат, чтобы вернуть данные к известному согласованному состоянию. Каждая транзакция продолжается до тех пор, пока она не завершается без ошибок и выполняется инструкция COMMIT TRANSACTION, чтобы внести изменения в базу данных. В случае возникновения ошибок все изменения удаляются с помощью инструкции ROLLBACK TRANSACTION.

Инструкция BEGIN TRANSACTION запускает локальную транзакцию для соединения, выполняющего эту инструкцию. В зависимости от текущих параметров уровня изоляции транзакции многие ресурсы, необходимые для поддержки выполняемых соединением инструкций языка Transact-SQL, блокируются транзакцией до ее завершения с помощью инструкций COMMIT TRANSACTION и ROLLBACK TRANSACTION. Транзакции, долгое время ожидающие обработки, могут препятствовать доступу других пользователей к заблокированным ресурсам и усечения транзакций в журнале.

Хотя инструкция BEGIN TRANSACTION запускает локальную транзакцию, она не записывается в журнал транзакций, пока приложение не выполнит действие, которое должно быть записано в журнал, например инструкцию INSERT, UPDATE или DELETE. Приложение может выполнять такие действия, как получение блокировки для защиты уровня изоляции транзакции инструкции SELECT, но ни одно действие не записывается в журнал, пока приложение не выполнит действие, изменяющее данные.

Присвоение имен нескольким транзакциям в последовательности вложенных транзакций мало влияет на транзакцию. Системой регистрируется только первое (самое внешнее) имя транзакции. Откат к другому имени (не считая допустимого имени точки сохранения) приводит к формированию ошибки. В момент возникновения такой ошибки на самом деле не выполняется

откат инструкций. Для этих инструкций выполняется откат только при откате внешней транзакции.

Локальная транзакция, запущенная инструкцией BEGIN TRANSACTION, повышается до распределенной транзакции, если до фиксации или отката инструкции выполняются следующие действия.

- Выполняется инструкция INSERT, DELETE или UPDATE, ссылающаяся на удаленную таблицу или связанный сервер. Инструкция INSERT, UPDATE или DELETE завершается неудачно, если поставщик OLE DB, который используется для доступа к связанному серверу, не поддерживает интерфейс **ITransactionJoin**.
- Вызывается удаленная хранимая процедура, если для параметра REMOTE_PROC_TRANSACTIONS установлено значение ON.

Локальная копия SQL Server становится контроллером транзакции и использует координатор распределенных транзакций Майкрософт (MS DTC) для управления распределенной транзакцией.

Транзакция может выполняться явно как распределенная с помощью инструкции BEGIN DISTRIBUTED TRANSACTION.

COMMIT TRANSACTION отмечает успешное завершение явной или неявной транзакции. Если значение параметра @@TRANCOUNT равно 1, то инструкция COMMIT TRANSACTION делает все изменения, выполненные с начала транзакции постоянной частью базы данных, освобождает ресурсы, удерживаемые транзакцией, и уменьшает значение параметра @@TRANCOUNT до 0. Если значение параметра @@TRANCOUNT больше 1, инструкция COMMIT TRANSACTION уменьшает значение параметра @@TRANCOUNT на 1 и оставляет транзакцию активной.

Обязанностью программиста на языке Transact-SQL является вызов инструкции COMMIT TRANSACTION только в том случае, когда все данные, относящиеся к транзакции, логически верны.

Если зафиксированная транзакция является распределенной транзакцией Transact-SQL, инструкция COMMIT TRANSACTION вызывает координатор MS DTC для использования двухфазного протокола фиксации на всех серверах, участвующих в транзакции. Если локальная транзакция охватывает две или более базы данных одного и того же экземпляра компонента Database Engine, то экземпляр использует встроенный двухэтапный алгоритм для фиксирования всех баз данных, вызванных в транзакции.

При использовании вложенных транзакций фиксация внутренних транзакций не освобождает ресурсы и не делает их изменения постоянными. Изменения данных становятся постоянными и освобождаются ресурсы только при фиксации внешней транзакции. Вызов каждой инструкции COMMIT TRANSACTION приводит к тому, что если значение параметра @@TRANCOUNT больше 1, то значение параметра @@TRANCOUNT просто уменьшается на 1. Когда значение параметра @@TRANCOUNT уменьшится до нуля, вся внешняя транзакция фиксируется. Так как аргумент transaction_name

не учитывается компонентом Database Engine, то вызов инструкции COMMIT TRANSACTION, содержащей ссылку на имя внешней транзакции, приводит к тому, что если существуют необработанные внутренние транзакции, то производится только уменьшение значения параметра @@TRANCOUNT на 1.

Вызов инструкции COMMIT TRANSACTION, когда значение параметра @@TRANCOUNT равно нулю, дает ошибку, так как нет соответствующей инструкции BEGIN TRANSACTION.

Нельзя произвести откат транзакции после вызова инструкции COMMIT TRANSACTION, так как измененные данные уже стали частью базы данных.

Компонент Database Engine в SQL Server 2000 и более поздних версиях увеличивает счетчик транзакций внутри выражения, только когда счетчик транзакций равен нулю при запуске инструкции. В версии 7.0 SQL Server счетчик транзакций всегда увеличивается вне зависимости от значения счетчика транзакций при запуске инструкции. Это может послужить причиной того, что значение, возвращаемое параметром @@TRANCOUNT в триггерах для SQL Server 2000 и более поздних версий, будет меньше, чем в SQL Server версии 7.0.

В SQL Server 2000 и более поздних версиях, если инструкции COMMIT TRANSACTION или COMMIT WORK выполняются в триггере и нет соответствующей явной или неявной инструкции BEGIN TRANSACTION при запуске триггера, пользователи могут наблюдать иное поведения, нежели в SQL Server версии 7.0. Помещение инструкций COMMIT TRANSACTION или COMMIT WORK в триггеры не рекомендуется.

Уровень изоляции транзакций

Транзакции указывают уровень изоляции, который определяет степень, до которой одна транзакция должна быть изолирована от изменений ресурса или данных, произведенных другими транзакциями. Уровни изоляции описаны с точки зрения того, какие из побочных эффектов параллелизма разрешены (например, «грязные» чтения или фантомные чтения).

Уровни изоляции транзакций контролируют следующие параметры.

- Применение и типы блокировки при чтении данных.
- Время удержания блокировок чтения.
- Использование операции чтения ссылок на строки, измененные другой транзакцией.
- Блокировка до тех пор, пока не будет снята монопольная блокировка строки.
- Извлечение зафиксированной версии строки, которая существовала в то время, когда началось выполнение инструкции или транзакции.
- Считывание незафиксированного изменения данных.

Выбор уровня изоляции транзакции не влияет на блокировки, примененные для защиты изменений данных. Транзакция всегда вызывает монопольную блокировку любых данных, которые она изменяет, и держит блокировку до тех пор, пока транзакция не завершится, независимо от уровня изоляции, установ-

ленного для транзакции. Для операций чтения уровни изоляции транзакций, в основном, определяют уровень защиты от эффектов изменений, сделанных другими транзакциями.

Более низкий уровень изоляции увеличивает возможность получения доступа к данным несколькими пользователями одновременно, но увеличивает число эффектов параллелизма (таких как «грязное» чтение или потерянные обновления), с которыми может столкнуться пользователь. Наоборот, более высокий уровень изоляции уменьшает число эффектов параллелизма, с которыми может столкнуться пользователь, но требует больше системных ресурсов и увеличивает шанс того, что одна транзакция блокирует другую. Выбор соответствующего уровня изоляции зависит от баланса между требованиями к целостности данных приложения и издержек каждого уровня изоляции.

Самый высокий уровень изоляции - SERIALIZABLE — изоляция упорядочиваемых транзакций — гарантирует, что транзакция получит в точности те же данные при каждой операции чтения, но достигается это применением уровня блокировки, при котором очень вероятно влияние на других пользователей в многопользовательских системах.

READ COMMITTED - указывает, что инструкции не могут считывать данные, которые были изменены, но не зафиксированы другими транзакциями. Это предотвращает грязное чтение.

REPEATABLE READ - указывает, что инструкции не могут считывать данные, которые были изменены, но еще не были зафиксированы другими транзакциями, и что никакие другие транзакции не могут изменять данные, которые были прочитаны текущей транзакции, пока текущая сделка будет завершена.

SNAPSHOT - указывает, что данные, считанные любым заявлением в сделке будут транзакционно соответствует версии данных, которые существовали в начале сделки.

Самый низкий уровень изоляции — READ UNCOMMITTED — может извлечь данные, которые были изменены, но не зафиксированы другой транзакцией. При изоляции уровня read uncommitted могут проявиться все эффекты параллелизма, но при таком уровне нет блокировки чтения или управления версиями, так что издержки минимальны.

Использование вложенных транзакций

Явные транзакции могут быть вложенными. Обычно это используется для поддержки транзакций в хранимых процедурах, которые могут быть вызваны, или из процесса, который уже находится в транзакции, или из процесса, у которого нет активной транзакции.

Пример показывает целевое использование вложенных транзакций. Процедура TransProc выполняет действия, независимо от режима транзакции. Если TransProc вызывается при активной транзакции, вложенные транзакции в TransProc игнорируются, а его INSERT фиксируются или откатываются. Если

TransProc выполняется процесс, который не имеет текущей транзакции, COMMIT TRANSACTION в конце процедуры выполняет INSERT.

SELECT в конце запроса показывает только строки 3 и 4 по- в таблице.

Пример:

```
CREATE TABLE TestTrans(Cola INT PRIMARY KEY,
        Colb CHAR(3) NOT NULL);
GO
CREATE PROCEDURE TransProc @PriKey INT, @CharCol CHAR(3) AS
BEGIN TRANSACTION InProc
INSERT INTO TestTrans VALUES (@PriKey, @CharCol)
INSERT INTO TestTrans VALUES (@PriKey + 1, @CharCol)
COMMIT TRANSACTION InProc;
GO
BEGIN TRANSACTION OutOfProc; /* Starts a transaction */
GO
EXEC TransProc 1, 'aaa';
GO
ROLLBACK TRANSACTION OutOfProc; /* Rolls back the outer
                                                    transaction */
GO
EXECUTE TransProc 3,'bbb';
GO
SELECT * FROM TestTrans;
GO
```

Результат:

```
Cola  Colb
------------
1                bb
2                bb
```

7. Запросы метаданных, XML и полнотекстовые индексы

7.1. Запросы метаданных

Метаданные

В большинстве проектах бизнес-аналитики, метаданные классифицируются по использованию, либо как деловые или технические метаданные. Бизнес - метаданные доступны для конечных пользователей, помогают объяснить источник или значение данных. Технические метаданные относится к метаданным, которые используются для поддержки функций, и, следовательно, представляют интерес в первую очередь для разработчиков и администраторов. Не-

которые примеры метаданных включают данные моделей, схем, таблиц, столбцов, индексы, кубы, отчеты

Чтобы помочь разработчику создавать и работать с метаданными, Analysis Services (SSAS) появилась новая функция под названием Универсальная модель измерений, или UDM. С помощью UDM, разработчик может создать единую, унифицированную логическую модель для OLAP и реляционной отчетности, а затем передать эту модель другим приложениям и системами посредством XML для анализа или OLE DB для OLAP. SSAS также имеет новую объектную модель называется AMO или анализ объектов.

Представления совместимости (Transact-SQL

Многие системные таблицы из предыдущих версий SQL Server в настоящее время реализованы в виде набора представлений. Эти представления известны как представления совместимости, и они предназначены только для обратной совместимости. Представления совместимости содержат метаданные, которые были доступны в SQL Server 2000. Однако представления совместимости не содержат метаданных, связанных с функциями, появившимися в SQL Server 2005 и более поздних версиях. Поэтому при использовании этих возможностей, таких как компонент Service Broker или секционирование, следует применять представления каталогов.

Еще одной причиной добавления обновлений в представления каталога является тот факт, что столбцы представлений совместимости, хранящие идентификаторы пользователей и типов, могут возвращать значение NULL или арифметические переполнения триггера. Это происходит потому, что можно создавать более 32 767 пользователей, групп и ролей, а также 32 767 типов данных. Например, если создать 32 768 пользователей, после чего выполнить следующий запрос: SELECT * FROM sys.sysusers. Этот запрос завершится ошибкой арифметического переполнения, если значение параметра ARITHABORT равно ON. Если значение параметра ARITHABORT равно OFF, столбец **uid** возвращает значение NULL.

Чтобы избежать таких проблем, рекомендуется использовать новые представления каталогов, которые могут содержать увеличившееся число идентификаторов пользователей и типов. В следующей таблице перечислены столбцы, которые могут подвергнуться переполнению.

Пример:

```
SELECT *
FROM sys.sysobjects;
```

Результат:

Name	Id	Xtype	Uid	Info	status
events	-414	V	4	0	0
event_notifications	-413	V	4	0	0
triggers	-412	V	4	0	0
procedures	-411	V	4	0	0
foreign_key_columns	-410	V	4	0	0

Пример показывает, как получить простой список объектов, которые существуют в текущей базе данных с помощью просмотра в режиме совместимости.

Представления системного каталога

Динамические приложения без жестко заданной необходимости работы с конкретным набором таблиц и представлений должны иметь механизм определения структуры и атрибутов объектов в любой базе данных, к которой они пытаются подключиться. Для этих приложений могут потребоваться следующие сведения.

- Количество и названия таблиц и представления в базе данных.
- Количество столбцов в таблице или представлении, а также имя, тип данных, масштаб и точность каждого столбца.
- Ограничения, определенные для таблицы.
- Индексы и ключи, определенные для таблицы.

Системный каталог предоставляет эти сведения для баз данных SQL Server. Ядром системных каталогов SQL Server является набор представлений, которые показывают метаданные, описывающие объекты в экземпляре SQL Server. Метаданные — это данные, описывающие атрибуты объектов в системе. Приложения с поддержкой SQL Server могут обращаться к этим сведениям в системных каталогах с помощью следующих элементов:

- Представления каталога (рекомендуется использовать именно этот способ доступа).
- Представления информационной схемы.
- Наборы строк схемы OLE DB.
- Функции работы с каталогами ODBC.
- Системные хранимые процедуры и функции.

Представления каталогов обеспечивают доступ к метаданным, которые хранятся во всех базах данных на сервере.

Рекомендуется использовать представления каталога для обращения к метаданным по следующим причинам:

- Все метаданные доступны в качестве представлений каталога.
- Представления каталога показывают метаданные в формате, который не зависит от реализации конкретной таблицы каталога, вследствие чего на них не влияют изменения в базовых таблицах каталога.
- Представления каталога являются наиболее эффективным способом доступа к основным метаданным сервера.
- Представления каталога — это стандартный интерфейс метаданных каталога. Они обеспечивают наиболее простой способ получения, преобразования и отображения настраиваемых форм метаданных.
- Имена представлений каталога и их столбцов являются описательными. Результаты запросов соответствуют ожидаемым результатам пользователей, не обладающих большими знаниями по функции, которая соответствует запрашиваемым метаданным.

Пример показывает, как получить список всех объектов в текущей базе данных и их свойства, используя представление sys.objects.

```
SELECT *
FROM sys.objects;
```

Результат:

name	**object_id**	**principal_id**	**schema_id**
sysrowsetcolumns	4	NULL	4
sysrowsets	5	NULL	4
sysallocunits	7	NULL	4
sysfiles1	8	NULL	4

Пример: представление sys.tables содержит информацию о каждой таблице в активной базе данных. Этот запрос возвращает имя и тип каждой таблицы.

```
SELECT name, type_desc
FROM sys.tables;
```

Результат:

Name	Type_desc
Product	USER_TABLE
Employee	USER_TABLE
Customer	USER_TABLE

Пример: представление sys.views предоставляет информацию обо всех представлениях в активной базе данных. Этот запрос возвращает имя, дата создания и последнего изменения для каждого представления.

```
SELECT name, create_date,
        modify_date
FROM sys.views;
```

Результат:

Name	Create_date	Modify_date
vwProducts	7/12/2007	7/12/2007
vwCustomer	7/14/2007	3/22/2008
vwProducts	7/12/2007	2/3/2008
vwOrders	7/12/2007	7/12/2007

Представления информационной схемы

Представление информационной схемы является одним из способов получения метаданных, которые предоставляет SQL Server.

Представления информационной схемы обеспечивают внутренний, системный, не зависящий от таблиц обзор метаданных SQL Server.

Представления информационной схемы позволяют приложениям правильно работать, несмотря на значительные изменения, внесенные в базовые системные таблицы. Представления информационной схемы, включенные в SQL Server, соответствуют стандартному определению ISO для INFORMATION_SCHEMA.

SQL Server поддерживает соглашения по трехкомпонентному именованию при ссылках на текущий сервер. Стандарт ISO также поддерживает согла-

шения по трехкомпонентному именованию. Однако имена, которые используются в обоих соглашениях, различаются. Представления информационной схемы определяются в специальной схеме с именем INFORMATION_SCHEMA. Эта схема содержится в любой базе данных. Каждое представление информационной схемы содержит метаданные для всех объектов, хранящихся в этой конкретной базе данных. В следующей таблице представлены связи между именами SQL Server и стандартными именами SQL.

Имя SQL Server	**Соответствует эквивалентному стандартному имени SQL**
База данных	Каталог
Схема	Схема
Объект	Объект
Определяемый пользователем тип данных (псевдоним)	Домен

Пример: представление INFORMATION_SCHEMA.COLUMNS предоставляет информацию о каждом столбце в базе данных. Запрос возвращает все имена столбцов для таблицы с именем "продукт".

```
SELECT COLUMN_NAME
FROM INFORMATION_SCHEMA.COLUMNS
WHERE TABLE_NAME = N'Product';
```

Результат:

COLUMN_NAME
Name
ProductNumber
MakeFlag
FinishedGoodsFlag

Пример: представление INFORMATION_SCHEMA.VIEWS предоставляет информацию о каждом представлении в базе данных. Запрос возвращает имена всех представлений в схеме «Продажи».

```
SELECT TABLE_NAME
FROM INFORMATION_SCHEMA.VIEWS
WHERE TABLE_SCHEMA = N‘Sales';
```

Результат:

TABLE_NAME
vIndividualCustomer
vPersonDemographics
vSalesPerson
vStoreWithContacts

Пример: представление INFORMATION_SCHEMA.TABLES предоставляет информацию обо всех таблицах, которые существуют в активной базе данных. Запрос возвращает имя таблиц в текущей схеме.

```
SELECT TABLE_NAME
FROM INFORMATION_SCHEMA.TABLES
WHERE TABLE_SCHEMA = N‘Person';
```

TABLE_NAME
AddressType
StateProvince
BusinessEntity
ContactType

Управление динамическими представлениями и функциями

Динамические представления похожи на системы каталогов в структуре. Разница в характере данных, которые они предоставляют. Динамические представления доступны для просмотра в SSMS в системных представлениях в обозревателе объектов.

Категория	Примеры	
	Представления	Функции
Execution	dm_exec_requests	dm_exec_sql_text

Index	dm_db_index_usage_stats	dm_db_missing_index_columns
I/O	dm_io_pending_io_requests	dm_io_virtual_file_stats
Operating System	dm_os_sys_info	

Пример показывает, как получить список команд, которые выполняются на сервере, и количество обращений к ним.

```
SELECT count(*) as Cnt, Command
FROM sys.dm_exec_requests
GROUP BY Command
```

Результат:

Cnt	Command
1	BRKR EVENT HNDLR
3	BRKR TASK
1	CHECKPOINT
5	FSAGENT TASK

Пример: функция dm_io_virtual_file_stats предоставляет информацию о состоянии и использовании баз данных SQL структуры файлового сервера.

```
SELECT database_id, num_of_reads, num_of_writes
FROM sys.dm_io_virtual_file_stats(DB_ID(N'AdventureWorks'), 2);
```

Результат:

database_id	num_of_reads	num_of_writes
7	8	7

Пример: dm_os_sys_info представление предоставляет информацию о сервере и операционной системе, которая поддерживает экземпляр SQL Server.

```
SELECT cpu_count, physical_memory_in_bytes, sqlserver_start_time
FROM sys.dm_os_sys_info
```

Результат:

cpu_count	physical_memory_in_bytes	sqlserver_start_time
1	960995328	2008-08-14 12:41:30.423

Системные хранимые процедуры и функции

Базу данных, файл, секцию и свойства файловой группы можно просматривать при помощи разных представлений каталогов, системных функций и системных хранимых процедур.

В следующей таблице приводятся представления каталогов, системные функции и системные хранимые процедуры, возвращающие сведения о базах данных, файлах и файловых группах.

Представления	Функции	Хранимые процедуры и другие инструкции
sys.databases	DATABASE_PRINCIPAL_ID	sp_databases
sys.database_files	DATABASEPROPERTYEX	sp_helpdb
sys.data_spaces	DB_ID	sp_helpfile
sys.filegroups	DB_NAME	sp_helpfilegroup
sys.allocation_units	FILE_ID	sp_spaceused
sys.master_files	FILE_IDEX	DBCC SQLPERF
sys.partitions	FILE_NAME	
sys.partition_functions	FILEGROUP_ID	
sys.partition_parameters	FILEGROUP_NAME	
sys.partition_range_values	FILEGROUPPROPERTY	
sys.partition_schemes	FILEPROPERTY	
sys.dm_db_partition_stats	fn_virtualfilestats	
sys.dm_db_file_space_usage (Transact-SQL) (только **tempdb**)		

Представление sys.dm_db_session_space_usage (только **tempdb**)		
sys.dm_db_task_space_usage (только **tempdb**)		

Если указываемая база данных недоступна, некоторые столбцы в представлении каталога sys.databases и свойства в функции DATABASEPROPERTYEX могут возвращать значение NULL.

Например, для возврата имени параметров сортировки в базе данных необходимо выполнить доступ к базе данных. Если база данных не находится в оперативном режиме или если параметр AUTO_CLOSE установлен в ON, имя параметров сортировки не может быть возвращено.

Пример показывает, как получить длину столбца таблицы в текущей базе данных, используются функции COLUMNPROPERTY и OBJECT_ID.

```
SELECT COLUMNPROPERTY( OBJECT_ID('Person.Contact'),
                       'LastName',
                       'PRECISION') AS 'Column Length';
```

Пример показывает, как получить список столбцов таблицы Department в схеме HumanResources.

```
EXEC sp_columns @table_name = N'Department',
                @table_owner = N'HumanResources';
```

7.2. Обзор XML

Основные сведения об использовании XML в SQL Server

XML это язык разметки, описывающий структуру данных, не содержит фиксированного набора элементов, как HTML.

Основа XML - тег.

Метки носят описательный характер.

Дизайнеры могут свободно создавать свои собственные XML-структуры и тэги для размещения данных.

XML стандарт не определяет структуру и содержание, определяет только правила.

Пример фрагмента:

```
<people>
  <person>
    <first_name>Robert</first_name>
    <last_name>Frost</last_name>
  </person>
  <person>
    <first_name>Frank</first_name>
    <last_name>Baker</last_name>
  </person>
</people>
```

Технические сценарии использования XML

XML используется в самых разнообразных технических сценариях.

1. Обмен структурированными данными между системами. В современном мире бизнеса распространены разнообразные информационные системы. Во многих случаях система одного типа должна передавать и принимать данные, содержащиеся в системе другого типа. XML независим от систем и может быть использован для обмена данными между различными системами.

2. Передача данных в интернете. XML представляет собой текстовый формат, так что данные в формате XML могут быть легко переданы через Интернет. Передача XML данных может быть обеспечена одними и теми же методами, которые используются для обеспечения передачи данных в формате HTML.

3. Кодирование документов. Информация хранится в различных форматах из различных систем и приложений. По хранение информации из одного приложения в открытом формате XML удобно с той точки зрения, что информация может быть использована в других приложениях и системах.

4.Преобразование данных. Данные многократно сохраняются системами и приложениями в собственном двоичном формате. При этом данные должны быть переданы в текстовый протокол или храниться в текстовом формате, они могут быть преобразованы в XML-формат.

Бизнес сценарии использования XML

Практически любая бизнес функция, которая включает в себя данные, может использовать XML.

1.Страховые выплаты. Компания автострахования предоставляет услуги в сети Интернет позволяя своим клиентам страховые услуги или ведет страховые выплаты через веб-сайт компании. Все обращения будут обработаны централизованной системой, которая находится на корпоративном сервере. После завершения обработки, система требует для хранения специальной информации, связанной с обращением формат XML.

Точные копии документов XML должны быть сохранены в системе для юридических целей. Этот сценарий показывает как можно использовать XML для управления контентом.

2.Обмен данными между производителем автомобилей и поставщиками запчастей. Автомобильный производитель взаимодействуют с несколькими поставщиками для закупки комплектующих, необходимых для компании. В настоящее время производитель получает счета-фактуры от поставщиков. Данные, соответствующие этим счетам затем вручную подаются в системы обработки счетов. В счете-фактуре система обработки хранит данные в реляционной форме. Производитель хочет автоматизировать обработку данных в системе обработки счетов. Этот сценарий является примером использования XML для интеграции бизнес-процессов.

3.Обмен данными между производителем автомобилей и поставщиками запчастей. Этот сценарий включает в себя автомобильного производителя, который взаимодействует с несколькими поставщиками комплектующими , как указано в предыдущем сценарии. Существующая система производителей не предоставляют возможность проверить состояние счета поставщикам, чтобы или получить копию платежного поручения от производителя. В настоящее время эта информация доступна для поставщиков только по телефону. Производитель автомобилей должен быть в состоянии передать эту информацию через интернет, так чтобы поставщики могли получать копию платежного поручения автоматически. Этот сценарий демонстрирует использование XML для интеграции бизнес-процессов.

4.Система управления контентом. Компания предоставляет информацию в области медицины, права и технологий для своих клиентов по различным каналам, включая Интернет, книги и CD-ROM. Компания хочет построить систему управления контентом, чтобы помочь доставить качественный контент для своих клиентов за меньшее время. Этот сценарий иллюстрирует использование XML для управления контентом.

5. Опрос клиентов. Компания предоставляет услуги бронирования авиабилетов в Интернете и проводит обследования для каждого сезона, чтобы определить наиболее популярные направления для клиентов на текущий сезон. Опросник используется для каждого сезона, отличается, и может изменяться в будущем. Компания будет анализировать информацию и результаты анализа будут использованы для разработки пакетов специальных предложений по путешествиям, которые будут удовлетворять потребности максимального числа клиентов. Этот сценарий может быть классифицирован как использование XML для управления контентом.

Реализация XML в SQL Server 2008

Microsoft SQL Server ® 2008 поддерживает несколько методов для запросов и анализа данных, сохраненных в формате XML. Реляционные данные могут быть преобразованы в XML с помощью XML для обработки в T-SQL.

Данные, которые уже существуют в формате XML, могут быть проанализированы и преобразованы в реляционные данные с помощью функции OpenXML. Кроме того, тип данных XML предоставляет методы для запроса, извлечения значений, и изменения данных, которые хранятся в формате XML.

XQuery является языком, который позволяет сделать запрос к структурированным или частично структурированным XML-данным. С поддержкой типа данных XML, представленных в Database Engine, документы могут быть сохранены в базе данных и запрашиваться при помощи XQuery. XQuery основан на существующих запросах XPath языка, добавлена поддержка для лучшей итерации, результатов сортировки, а также возможность построить необходимые XML преобразования.

Перед созданием типизированных XML переменных, параметров или столбцов, сначала требуется зарегистрировать коллекцию XML-схем с помощью CREATE XML SCHEMA COLLECTION (Transact-SQL). Затем можно связать коллекцию XML-схем с переменными, параметрами или столбцами XML тип данных.

Тип данных XML

В версии SQL Server 2005 был введен тип данных **xml**, позволяющий хранить XML-документы и их фрагменты в базе данных SQL Server. Тип данных **xml** — это встроенный в SQL Server тип данных, напоминающий другие встроенные типы данных, такие как **int** и **varchar**. Как и другие встроенные типы данных, тип данных **xml** можно использовать как тип столбца при создании таблицы, как тип переменной, параметра, тип возвращаемого функцией значения, а также в инструкциях CAST и CONVERT.

Тип данных XML поддерживает пять методов для работы с данными.

Query	Использует XQuery для ввода и возвращает экземпляр типа данных XML
Value	Возвращает значение типа SQL из XML выражения
Exists	Определяет возвращает ли XQuery выражение результат
Modify	Используется для добавления, модификации и удаления XML данных
Nodes	Разделяет XML на несколько строк

7.3.Запросы XML данных

Использование FOR для генерации XML

XML может быть использован для создания XML-представления любого набора строк.

XML поддерживает расширения для обеспечения контроля за полученной XML-структурой.

Существует три режима работы и управления форматом результатов XML - RAW, AUTO и EXPLICIT .

Корень может быть использован для обеспечения перебора по пути от корневого узла до результирующего узла дерева. В XML единый элемент верхнего уровня, можно указать имя корневого элемента для создания. Значение по умолчанию "ROOT".

Режим AUTO- возвращает результаты запроса в виде простого вложенного XML дерева. Каждая таблица в выражении FROM, для которой, по крайней мере, один столбец перечисленный в SELECT представлен в виде XML-элемента. Столбцы, перечисленные в SELECT, сопоставляются с соответствующими атрибутами элемента.

Режим RAW- принимает результат запроса и преобразует каждую строку в наборе результатов в XML-элемент, который имеет общий идентификатор row, как элемент тега. Можно дополнительно указать имя для элемента строки, когда используется эта директива, в результате XML будет использовать указанный элемент в качестве имени строки, который создается для каждой строки.

Режим EXPLICIT - указывает, что форма полученного XML дерева определена в явном виде. При использовании этого режима, запросы должны быть написаны особым образом.

Пример, информация о клиентах извлекается из трех различных таблиц - Cust, SalesOrderHeader, OrderHeader. Извлекаются значения идентификатора клиента, ID заказов, состояния и типа клиента и с помощью предложения FOR XML результат преобразуется в автоматическом режиме.

```
SELECT Cust.CustomerID,
    OrderHeader.CustomerID,
    OrderHeader.SalesOrderID,
    OrderHeader.Status,
    Cust.CustomerType
FROM Sales.Customer Cust, Sales.SalesOrderHeader OrderHeader
WHERE Cust.CustomerID = OrderHeader.CustomerID
ORDER BY Cust.CustomerID
FOR XML AUTO {RAW('ElementName') | EXPLICIT}
```

Результат:

```
<Cust CustomerID="1" CustomerType="S">
  <OrderHeader CustomerID="1" SalesOrderID="43860" Status="5" />
  <OrderHeader CustomerID="1" SalesOrderID="44501" Status="5" />
  <OrderHeader CustomerID="1" SalesOrderID="45283" Status="5" />
  <OrderHeader CustomerID="1" SalesOrderID="46042" Status="5" />
</Cust>
```

Запрос XML данных с помощью OpenXML

OpenXML используется для разбора входных данных XML и преобразования их в реляционный формат, с которым легче работать в SQL Server.

В Transact-SQL ключевое слово OPENXML представляет собой набор строк в памяти, который похож на таблицу или представление.

Для подготовки чтения XML-данных, перед использованием OpenXML должна быть вызвана системная хранимая процедура sp_xml_preparedocument.

OpenXML позволяет выбирать строки и столбцы по XPath путям как переменные.

Пример строки XML в локальной переменной, содержащий информацию об имени, состоящего из первого и последнего атрибута имени.

Добавлены только 3 строки, после того, как XML строка создана, она передается в виде параметра в sp_xml_preparedocument процедуру, а затем используется в запросе на выборку использованием OPENXML. После того как запрос был выполнен, вызывается хранимая процедура sp_xml_removedocument, чтобы очистить память.

```
DECLARE @xml_text VARCHAR(4000), @i INT
SELECT @xml_text =
'<root><person LastName="White" FirstName="Johnson"/>
<person LastName="Green" FirstName="Marjorie"/>
<person LastName="Carson" FirstName="Cheryl"/></root>'
EXEC sp_xml_preparedocument @i OUTPUT, @xml_text
SELECT * FROM
   OPENXML(@i, '/root/person') WITH (LastName nvarchar(50),
                                     FirstName nvarchar(50))
EXEC sp_xml_removedocument @i
```

Результат:

LastName	FirstName
White	Johnson
Green	Marjorie
Carson	Cheryl

Запрос XML данных с помощью XQuery

XQuery является языком запросов, разработан и создан для выполнения запросов к данным, хранящимся в формате XML.

XQuery определяет синтаксис FLWOR.

FLWOR является аббревиатурой -for, let, where, order by, и return.

Пример, таблица ProductModel содержит столбец типа XML, с помощью XQuery выполняется запрос для выбора данных.

```
SELECT Instructions.query('
   declare namespace AWMI="http://schemas.microsoft.com
/sqlserver/2004/07/adventureworks/ProductModelManuInstructions";
      for $T in //AWMI:tool
        let $L := //AWMI:Location[.//AWMI:tool[.=data($T)]]
      return
       <tool desc="{data($T)}" Locations="{data($L/@LocationID)}"/>
') as Result
FROM Production.ProductModel
where ProductModelID=7
```

Результат:

```
<tool desc=“hammer” Locations=“30”/>
```

Генерация базовых отчетов XML

Поставщик XML данных поставляет данные для Reporting Services в формате XML. Содержание XML может быть выбрано непосредственно в запросе и разработчик может создавать запросы и выбирать данные динамически при построении отчета.

XML-содержимое можно также получить непосредственно из URL или XML поставщик данных может запросить веб-службы непосредственно путем размещения XML-структуры непосредственно на SOAP.

7.4. Обзор полнотекстовых индексов

Полнотекстовые индексы

Полнотекстовый индекс представляет собой особый тип маркера функционального индекса построеный и поддерживаемый Microsoft на основе полнотекстового поиска для SQL Server ® (MSFTESQL).

Достаточно часто требуется использование полнотекстового индекса в том числе:

Предоставление данных для расширенного поиска по веб-сайту.

Разрешение нечеткого поиска описаний продуктов.

Разрешение шаблона поиска адресов клиентов.

Заполнение полнотекстовых индексов

Процесс создания и поддержания полнотекстовых индексов называется заполнением.

Microsoft поддерживает следующие типы заполнения:

Полное заполнение (Full Population). Данный тип заполнения индекса осуществляется при его создании. Предполагается, что в дальнейшем его поддержание осуществляется другими типами заполнения.

Заполнение на основе отслеживания изменений (Change Tracking Based Population). Для тех индексированных таблиц, где установлен данный тип заполнения, SQL Server поддерживает запись измененных строк. Записанные изменения затем переносятся в полнотекстовый индекс. Замечу, что данный тип заполнения будет работать, если предварительно уже было произведено заполнение индекса.

Инкрементное заполнение на основе версии строки (Incremental Timestamp Based Population). Для того чтобы использовать данный тип заполнения, в таблице должен быть столбец с типом timestamp. Для запуска этого типа заполнения используем пункт контекстного меню Full-Text Index | Start Incremental Population, щелкнув предварительно по строке с именем таблицы в разделе Tables в окне Object Explorer.

Реализация полнотекстовых индексов в SQL Server 2008

Для реализации полнотекстовых индексов требуется выполнить следующие действия:

1. Определить таблицы и столбцы, которые требуют полнотекстового индексирования в соответствии с бизнес- задачами
2. Включить полнотекстовую индексацию в базе данных

Используйте sp_fulltext_database ‘enable’

3. Создайть полнотекстовый индекс для необходимых таблиц
   Используйте sp_fulltext_table
4. Добавить необходимые столбцы к индексу
   Используйте sp_fulltext_column
5. Активизировать и установить опции совокупности для индекса
   Используйте sp_fulltext_table
6. Разработать запросы с использованием полнотекстовых функций запроса и предикатов в соответствии с требованиями бизнес – задач

7.5. Запросы с использованием полнотекстовых индексов

Обзор полнотекстового поиска

Microsoft SQL Server ® 2008 содержит функции, необходимые для выполнения полнотекстовых запросов к простым символьным данным в таблицах SQL Server .

Полнотекстовые запросы могут включать слова и фразы или несколько форм слова или фразы, и позволяют быстро и гибко индексировать ключевые слова запроса на основе текстовых данных, хранящихся в базе данных.

Большинство опытных пользователей SQL Server знакомы с поиска текстовых данных с помощью ключевого слова LIKE, однако, когда вам нужно выйти за пределы стандартных предикатов SQL существует возможность использовать полнотекстовый поиск .

В SQL Server реализован ряд предикатов для выполнения полнотекстового поиска.

В SQL Server в полнотекстовых предикатах CONTAINS или FREETEXT при выполнении запросов к связанным серверам можно использовать четырехкомпонентные имена.

Предикат CONTAINS

CONTAINS может производить поиск:

- слова или фразы;
- префикса слова или фразы;
- слова около другого слова;
- слова, флективно сформированного из другого (например, слово «drive» является флективной основой слов «drives», «drove», «driving» и «driven»);
- слова, которое является синонимом другого слова, с использованием тезауруса (например, слово «metal» может иметь синоним «aluminum» и «steel»).

Примеры:

```
SELECT Name
FROM Production.Product
WHERE CONTAINS(Name, ' "Chain*" ');
```

```
CONTAINS(Name, ' "Mountain" OR "Road" ')
CONTAINS(Description, 'bike NEAR performance')
CONTAINS(Description, ' FORMSOF (INFLECTIONAL, ride) ')
CONTAINS(Description, 'ISABOUT (performance weight (.8), comfortable weight (.4), smooth weight (.2) )' )
```

Предикат FREETEXT

Этот предикат используется в предложении WHERE для поиска в столбцах, содержащих символьные типы данных, тех значений, которые соответствуют условию поиска по смыслу, а не написанию. Когда используется предикат FREETEXT, ядро полнотекстовых запросов автоматически выполняет описанные далее действия над строкой freetext_string, присваивает вес каждому терму, а затем ищет совпадения.

- Разбивает строку на отдельные слова согласно границам слов (пословное разбиение).
- Формирует словоформы (а также производит выделение основы слова).
- Определяет список расширений или замен для термов на основании совпадений в тезаурусе.

Пример:

```
SELECT Title
FROM Production.Document
WHERE FREETEXT (Document, 'vital safety components' );
```

Результаты:

```
Возврат слов: “vital”, “safety”, “components”
Выделение основы (слова): “vital”, “safe”, “safety”, “components”
Список расширений: “vital”, “important”, “safe”, “safety”, “components”, “parts”
```

Общие сведения о полнотекстовых предикатах и функциях

Полнотекстовые запросы используют полнотекстовые предикаты (CONTAINS и FREETEXT) и функции (CONTAINSTABLE и FREETEXTTABLE). Эти предикаты и функции поддерживают расширенный синтаксис Transact-SQL, который поддерживает разнообразные формы выражений запроса. Для создания полнотекстовых запросов нужно знать, когда и как использовать полнотекстовые предикаты и функции. В этом разделе описаны предикаты и функции, рассматриваются общие возможности предиката CONTAINS и функции CONTAINSTABLE.

Комбинация полнотекстового поиска и предикатов T-SQL

Предикаты CONTAINS и FREETEXT можно сочетать с любыми другими предикатами языка Transact-SQL, например LIKE и BETWEEN. Их также можно использовать во вложенных запросах.

Пример:

```
SELECT Product.ProductDescriptionID, Product.Description, Keys.Rank
FROM Production.ProductDescription Product
   INNER JOIN FREETEXTTABLE (Production.ProductDescription, [Description],
      'safety',LANGUAGE 'English',2) AS Keys
      ON Product.ProductDescriptionID = Keys.[KEY]
WHERE Product.QuantityAvailable > 0;
```

Результат:

ProductDescriptionID	Description	Rank
513	All-occasion value bike with our basic comfort and safety features. Offers wider, more stable tires for a ride around town or weekend trip.	134
594	Travel in style and comfort. Designed for maximum comfort and safety. Wide gear range takes on all hills. High-tech aluminum alloy construction provides durability without added weight.	67

8. Использование программных объектов для получения данных

8.1.Представления

Microsoft SQL Server ® поддерживает возможность создания виртуальных таблиц, известных как представление.

Представление можно рассматривать либо как виртуальную таблицу или хранимый запрос. Данные доступные через представление не хранятся в базе данных как отдельный объект, в базе данных хранится SELECT, который формирует виртуальную таблицу, возвращаемую в представлении.

Пользователь может использовать виртуальную таблицу, ссылаясь на представление, так же, как таблицу.

Представление используется, чтобы:

• Ограничить доступ пользователей к определенным строкам в таблице. Например, позволяющим сотруднику видеть только те строки или записи, которые необходимы для его работы.

• Ограничить доступ пользователей к определенным столбцам. Например, позволяющие сотрудникам, которые не работают с фондом заработной платы, видеть название, офиса, рабочего телефона, а также отдел сотрудника, но не позволять им видеть столбцы с информацией заработной платы или личной информации.

• Отображать столбцы из нескольких таблиц, так чтобы они выглядели как одна таблица.

• Предоставлять совокупную информацию. Например, представить сумму столбца, максимальное или минимальное значение из столбца.

Создание и управление представлением

```
[CREATE|ALTER] VIEW HumanResources.vEmployee
AS
BEGIN
SELECT EmployeeID, FirstName, LastName, EmailAddress
FROM HumanResources.Employee
END
```

Создает виртуальную таблицу, содержимое которой (столбцы и строки) определяется запросом. Используйте эту инструкцию для создания представления данных, содержащихся в одной или более таблицах базы данных.

Представление может быть создано только в текущей базе данных и может иметь не более 1024 столбцов.

Вы можете изменить определение существующего вида с помощью инструкции ALTER VIEW.

Вы также можете создавать индексированные представления.

Удалить одно или несколько представлений из текущей базы данных с помощью инструкции DROP VIEW.

После того, как представление создано оно может быть запрошено как обычная таблица. При выполнении запроса к представлению, Database Engine проверяет, что все объекты базы данных, указанные в представлении существуют и что они действуют в контексте представления. Database Engine также проверяет, что операторы на изменение данных не нарушают целостности данных правил или других норм просмотра таких, как изменение вычисляемого поля.

При попытке запросить представление, которое зависит от таблицы или представления, которые не существуют, Database Engine выдает сообщение об ошибке. Однако, если новые таблицы или представления, созданы с той же структурой, представление снова становится пригодным для использования.

Ограничения при создании представлений

При создании представления нужно учитывать следующие факторы:

- Количество ссылок на столбцы в представлении не может быть больше 1024
- Не может быть использовано в инструкции CREATE VIEW
- Не может быть использовано в представлении, встроенных функциях, производных таблицах и вложенных запросах
- Не может быть использовано в SELECT при обращении к представлению
- Не может ссылаться на представление
- CREATE VIEW должно быть только в одном пакете
- Может быть использовано при определении представления, если не указан пункт SCHEMABINDING

Ограничения на изменение данных с помощью представлений

При реализации изменения данных с помощью представления следует учитывать два виде ограничений.

Ограничения на написание инструкций, которые изменяют данные.

- Инструкции должны изменять столбцы только одной базовой таблицы
- Следуйте правилам при использовании WITH CHECK OPTION
- В INSERT необходимо указать значения для всех необнуляемых столбцов

Ограничения на столбцы, которые изменяют данные:

- Столбцы таблицы должны иметь прямые ссылки
- Используйте триггер INSTEAD OF
- Не могут использоваться GROUP BY, HAVING, и DISTINCT.
- Данные должны придерживаться ограничений на изменение столбцов

Индексированные представления

Перед созданием кластеризованного индекса для представления оно должно удовлетворять следующим требованиям.

- При выполнении инструкции CREATE VIEW параметры ANSI_NULLS и QUOTED_IDENTIFIER должны быть установлены в ON.
- При создании инструкцией CREATE TABLE таблиц, на которые ссылается представление, параметр ANSI_NULLS должен быть установлен в ON.
- Представление не должно ссылаться ни на какие другие представления, ссылки должны указывать только на базовые таблицы.
- Все базовые таблицы, на которые ссылается представление, должны находиться в той же базе данных, что и представление, и иметь того же владельца, что и представление.
- Представление должно быть создано с параметром SCHEMABINDING. Это позволяет привязать представление к схеме базовых таблиц.
- Пользовательские функции, на которые ссылается представление, должны быть созданы с параметром SCHEMABINDING.
- Таблицы и пользовательские функции, на которые ссылается представление, должны указываться по именам из двух элементов. Имена из одного, трех или четырех элементов недопустимы.
- Все функции, на которые ссылаются выражения в представлении, должны быть детерминированными. Определить, является ли пользовательская функция детерминированной, можно по свойству **IsDeterministic** через функцию OBJECTPROPERTY.

Установка параметра столбца **large_value_types_out_of_row** в индексированном представлении наследуется от установки соответствующего столбца базовой таблицы. Это значение задается при помощи хранимой процедуры sp_tableoption. Для столбцов, созданных из выражений, установкой по умолчанию является 0.

После создания кластеризованного индекса любое соединение, которое пытается изменить базовые данные представления, должно иметь такие же установки параметров, какие необходимы для создания индекса. Если соединение, выполняющее инструкцию, имеет неверные установки параметров, SQL Server выдает ошибку и выполняет откат инструкций INSERT, UPDATE или DELETE, которые выполняются в отношении результирующего набора представления.

При удалении представления удаляются также и все его индексы. При удалении кластеризованного индекса удаляются все некластеризованные индексы и автоматически созданные для представления статистики. Статистики, созданные пользователем, сохраняются. Некластеризованные индексы могут удаляться по отдельности. При удалении кластеризованного индекса представ-

ления удаляется сохраненный результирующий набор, и оптимизатор снова начинает работать с ним, как с обычным представлением.

Хотя в инструкции CREATE UNIQUE CLUSTERED INDEX задаются только столбцы, образующие кластеризованный индексный ключ, в базе данных хранится весь результирующий набор представления. Как и в кластеризованном индексе базовой таблицы, структура сбалансированного дерева кластеризованного индекса содержит только ключевые столбцы, а в результирующем наборе представления строки данных содержат все столбцы.

Если необходимо добавить индексы к представлениям в существующей системе, то необходимо привязать к схеме представление, для которого создается индекс. Это можно сделать следующим образом.

- Удалить и вновь создать представление с указанием WITH SCHEMABINDING.
- Создать второе представление с тем же текстом, что и в существующем представлении, но под другим именем. Оптимизатор пользуется индексами нового представления даже в том случае, если в запросе предложение FROM на него непосредственно не ссылается.

Новое представление должно удовлетворять всем требованиям, предъявляемым к индексированным представлениям. Для этого может понадобиться изменить владельца представления и всех его базовых таблиц так, чтобы все они принадлежали одному пользователю.

Пример:

```
CREATE VIEW vwDiscount WITH SCHEMABINDING AS
SELECT SUM(UnitPrice*OrderQty)AS SumPrice,
SUM(UnitPrice*OrderQty*(1.00-UnitPriceDiscount))AS SumDiscountPrice,
SUM(UnitPrice*OrderQty*UnitPriceDiscount)AS SumDiscountPrice2,
COUNT_BIG(*) AS Count, ProductID
FROM Sales.SalesOrderDetail
GROUP BY ProductID
GO
CREATE UNIQUE CLUSTERED INDEX vwDiscountInd ON vwDiscount
(ProductID)
```

Представление отображает выбор нескольких вычисляемых столбцов, а также столбец ProductID и набор результатов, сгруппированных по ProductID. Результатом является то, что одна строка будет выводиться для каждого ProductID и для каждого будет вычислять общий объем продаж, общий объем скидок, а общая стоимость продаж для каждого продукта. Группировка по ProductID важна, чтобы ProductID столбец в представлении был уникальным для индекса.

CREATE UNIQUE INDEX CLUSTERED - используется для создания индекса в определенном представлении.

Секционированные представления

Секционированные представления позволяют разделить данные из большой таблицы на несколько таблиц-элементов меньшего размера. Данные распределяются по секциям между таблицами-элементами в зависимости от значений, содержащихся в одном из столбцов. Диапазоны данных для каждой из таблиц-элементов определяются в ограничении CHECK, устанавливаемом для столбца, по которому выполняется секционирование. Затем определяется представление, использующее инструкцию UNION ALL для объединения выборок из всех таблиц-элементов в единый результирующий набор. Когда инструкция SELECT, ссылающаяся на представление, содержит условие поиска по столбцу секционирования, оптимизатор запросов использует определение ограничений CHECK, чтобы определить, какие из таблиц-элементов содержат соответствующие запросу строки.

Пример:

```
CREATE TABLE May1998sales
(OrderID INT,
 CustomerID INT NOT NULL,
 OrderDate DATETIME NULL
CHECK (DATEPART(yy, OrderDate) = 1998),
 OrderMonth INT
CHECK (OrderMonth = 5),
 DeliveryDate DATETIME NULL
CONSTRAINT OrderIDMonth PRIMARY KEY(OrderID, OrderMonth)
)
```

```
CREATE VIEW Year1998Sales
AS
SELECT * FROM Jan1998Sales
UNION ALL
SELECT * FROM Feb1998Sales
UNION ALL
SELECT * FROM Mar1998Sales
UNION ALL
SELECT * FROM Apr1998Sales
UNION ALL
SELECT * FROM May1998Sales
UNION ALL
```

```
SELECT * FROM Jun1998Sales
UNION ALL
SELECT * FROM Jul1998Sales
UNION ALL
…
```

UNION ALL используется для объединения строк секционированной таблицы в представлении. В результате, если представление запрашивается без всяких условий, все строки из всех разбитых на разделы таблиц будут возвращены как один результирующий набор. Однако, если используется состояние даты заказа, то оптимизатор запросов будет использовать проверку ограничений секционированной таблицы, чтобы определить, какая таблица содержит строки, которые будут получены в представлении.

8.2. Пользовательские функции

Функции создаваемые пользователем

Подобно функциям в языках программирования, определяемые пользователем функции MicrosoftSQL Server, являются подпрограммами, которые принимают параметры, выполняют действия, такие как сложные вычисления, а затем возвращают результат этих действий в виде значения. Возвращаемое значение может быть либо единичным скалярным значением, либо результирующим набором.

Определяемые пользователем функции SQL Server предоставляют следующие преимущества.

- Делают возможным модульное программирование.

Можно, однажды создав функцию, сохранить ее в базе данных, а затем любое число раз вызывать из своей программы. Определяемые пользователем функции могут быть изменены независимо от исходного кода программы.

- Позволяют ускорить выполнение.

Как и хранимые процедуры, определяемые пользователем, функции Transact-SQL снижают стоимость компиляции кода Transact-SQL, кэшируя и повторно используя планы выполнения. Это означает, что для определяемых пользователем функций нет необходимости выполнять повторный синтаксический анализ и оптимизацию при каждом вызове, что значительно ускоряет их выполнение.

Функции CLR предоставляют значительный выигрыш в производительности по сравнению с функциями Transact-SQL для вычислительных задач, работы со строками и бизнес-логикой. Функции Transact-SQL лучше приспособлены для логики доступа к данным.

- Позволяют уменьшить сетевой трафик.

Операция, которая фильтрует данные на основе какого-нибудь сложного ограничения и не может быть выражена одним скалярным выражением, может

быть реализована в виде функции. Ее можно вызвать из предложения WHERE, чтобы уменьшить число строк, возвращаемых клиенту.

Компоненты определяемой пользователем функции

Определяемые пользователем функции могут быть написаны на языке Transact-SQL или на любом языке программирования .NET.

Любая определяемая пользователем функция состоит из двух частей: заголовка и текста. Функция принимает нуль и более параметров и возвращает либо скалярное значение, либо таблицу.

Заголовок определяет:

- имя функции с необязательным именем схемы или владельца;
- имя и тип данных входного аргумента;
- параметры, применимые к входному аргументу;
- тип данных возвращаемого значения и необязательное имя;
- параметры, применимые к возвращаемому значению.

Текст определяет действие или логику, которую выполняет функция. Он может содержать:

- одну или несколько инструкций Transact-SQL, реализующих логику функции;
- ссылку на сборку .NET.

Создание и управление пользовательскими функциями

Функция представляет собой подпрограмму Transact-SQL или среды CLR, которая возвращает значение. Пользовательские функции бывают скалярными или возвращающими табличное значение. Функция является скалярной, если в предложении RETURNS указан один из скалярных типов данных. Скалярные функции могут состоять из нескольких инструкций Transact-SQL. Функция является возвращающей табличное значение, если в предложении RETURNS указывается ключевое слово TABLE. В зависимости от того, каким образом определено тело функции, функции, возвращающие табличное значение, подразделяются на встроенные функции и функции из нескольких инструкций.

Пример:

```
[CREATE|ALTER] FUNCTION fEmployeeEmail(@ID int)
RETURNS varchar(50)
AS
BEGIN
DECLARE @email varchar(50)
SELECT @email = EmailAddress
FROM HumanResources.Employee
WHERE EmployeeID = @ID
RETURN @email
END
DROP FUNCTION fEmployeeEmail
```

Пример создания пользовательской функции, возвращающей табличное значение:

```
CREATE FUNCTION fEmployeeByGender(@Gender nchar(1))
RETURNS table
AS
BEGIN
RETURN (SELECT *
      FROM HumanResources.Employee
      WHERE Gender = @Gender)
END
```

```
SELECT * FROM fEmployeeByGender('F')
```

Ограничения при создании пользовательских функций

Пользовательской функции хранятся в виде объектов базы данных и обеспечивают повторно используемый код, который можно использовать поразному. Тем не менее, существуют ограничения при создании пользовательских функций.

Вы не можете использовать пользовательские функции для:

• Обновления данных. Вместо этого вы можете использовать хранимые процедуры.

• Определения или создания новых объектов в базе данных. Все объекты, на которые ссылается функция, за исключением скалярных типов, должны быть созданы ранее.

• Выполнения операций.

Категории функций:

В SQL Server ® 2008, функции могут быть разделены на две категории: детерминированные и недетерминированные. Функция является детерминированной, если для конкретного набора входных значений и состояния базы данных, функция всегда возвращает те же результаты. Недетерминированных функция может возвращать разные результаты при неоднократном вызове с тем же набором входных значений.

Например, функция GETDATE () является недетерминированной.

Функции, которые являются недетерминированными, не могут быть использованы в пользовательских функциях, индексированных представлениях, индексированных вычисляемых столбцах, или в вычисляемых столбцах.

Вопросы производительности при использовании пользовательских функций

В то время как определяемые пользователем функции могут уменьшить количество общего кода и могут быть использованы как часть общих функций библиотеки.

Не следует использовать пользовательские функции, если страдает производительность.

Например, предположим, что существует определенная пользователем функция, которая выполняет SELECT, и занимает одну секунду. Если эта пользовательская функция используется в SELECT, при вызове, и будет выполнена для каждой строки, время основного запроса, необходимое для выполнения может увеличиться резко в зависимости от таких факторов, как количество строк и типы индексов. Перед использованием определяемых пользователем функций в подобной ситуации, необходимо тщательно взвесить все варианты и сделать тестирования производительности.

8.3 Хранимые процедуры

Хранимая процедура — это сохраненная коллекция инструкций языка Transact-SQL или ссылка на метод среды CLR платформы Microsoft .NET Framework, которая может принимать и возвращать предоставленные пользователем параметры. Процедуры можно создавать для постоянного использования, для временного использования в одном сеансе (локальная временная процедура) или для временного использования во всех сеансах (глобальная временная процедура).

Хранимые процедуры могут выполняться автоматически при запуске экземпляра SQL Server

Стандартный максимальный размер хранимой процедуры не установлен.

Пользовательская хранимая процедура может быть создана только в текущей базе данных. Временные процедуры представляют собой исключение из этого правила, потому что они всегда создаются в базе данных **tempdb**. Если имя схемы не указано, используется схема по умолчанию, связанная с пользователем, который создает процедуру.

Инструкцию CREATE PROCEDURE нельзя объединять с другими инструкциями Transact-SQL в одном пакете.

По умолчанию параметры могут принимать значения NULL. Если параметр, имеющий значение NULL, используется в инструкции CREATE TABLE или ALTER TABLE при обращении к столбцу, не поддерживающему значения NULL, то компонент Database Engine возвращает ошибку. Чтобы предотвратить передачу значений NULL столбцу, который их не поддерживает, следует реализовать в процедуре соответствующую логику или передать столбцу значение по умолчанию при помощи ключевого слова DEFAULT инструкции CREATE TABLE или ALTER TABLE.

Для каждого столбца во временной таблице рекомендуется явно указывать атрибут NULL или NOT NULL. Если атрибуты NULL или NOT NULL не

указаны в инструкции CREATE TABLE или ALTER TABLE, то способ назначения этих атрибутов столбцам компонентом Database Engine определяется параметрами ANSI_DFLT_ON и ANSI_DFLT_OFF. Если в контексте соединения выполняется хранимая процедура с настройками этих параметров, отличными от настроек соединения, в котором была создана процедура, столбцы таблицы, созданной для второго соединения, могут отличаться по признаку поддержки допустимости значений NULL и работать иначе. Если атрибут NULL или NOT NULL явно задан для каждого столбца, временные таблицы создаются с одним и тем же признаком поддержки допустимости значений NULL во всех соединениях, в которых выполняется хранимая процедура.

Пример:

```
CREATE PROCEDURE HumanResources.usp_GetEmployeesName
@NamePrefix char(1)
AS
BEGIN
SELECT BusinessEntityID, FirstName, LastName, EmailAddress
FROM HumanResources.vEmployee
WHERE FirstName LIKE @NamePrefix + '%'
ORDER BY FirstName
END
```

```
EXECUTE HumanResources.usp_GetEmployeesName 'A‘
```

Использование хранимых процедур

При использовании хранимых процедур рекомендуется придерживаться следующих правил:

- Используйте WITH ENCRYPTION для того, чтобы скрыть источник выполнения процедуры
- Используйте WITH RECOMPILE для перекомпиляции при каждом вызове
- Проверьте все входные параметры
- Избегайте задавать строки, базирующиеся на SQL в процедуре для снижения риска инъекций SQL
- Экономно используйте курсоры.

8.4. Триггеры

Триггер — это особая разновидность хранимой процедуры, выполняемая автоматически при возникновении события на сервере базы данных. Триггеры языка обработки данных выполняются по событиям, вызванным попыткой пользователя изменить данные с помощью языка обработки данных. События-

ми DML являются процедуры INSERT, UPDATE или DELETE, применяемые к таблице или представлению.

Триггеры DDL срабатывают в ответ на ряд событий языка определения данных (DDL). Эти события прежде всего соответствуют инструкциям Transact-SQL CREATE, ALTER, DROP и некоторым системным хранимым процедурам, которые выполняют схожие с DDL операции. Триггеры входа могут срабатывать в ответ на событие LOGON, возникающее при установке пользовательских сеансов. Триггеры могут быть созданы непосредственно из инструкций Transact-SQL или из методов сборок, созданных в среде выполнения CLR платформы Microsoft.NET Framework, и переданы экземпляру SQL Server. SQL Server допускает создание нескольких триггеров для любой указанной инструкции.

Триггеры DML часто используются для соблюдения бизнес-правил и целостности данных. В SQL Server декларативное ограничение ссылочной целостности обеспечивается инструкциями ALTER TABLE и CREATE TABLE. Однако декларативное ограничение ссылочной целостности не обеспечивает ссылочную целостность между базами данных. Ограничение ссылочной целостности подразумевает выполнение правил связи между первичными и внешними ключами таблиц. Для обеспечения ограничений ссылочной целостности используйте в инструкциях ALTER TABLE и CREATE TABLE ограничения PRIMARY KEY и FOREIGN KEY. Если ограничения распространяются на таблицу триггера, они проверяются после срабатывания триггера INSTEAD OF и до выполнения триггера AFTER. В случае нарушения ограничения выполняется откат действий триггера INSTEAD OF, и триггер AFTER не срабатывает.

Пример:

```
CREATE TRIGGER Sales.trigCurrency
  ON  Sales.Currency
  AFTER INSERT
AS
BEGIN
        DECLARE @name nvarchar(50)
        SELECT @name = Name
        FROM inserted
        IF len(@name) < 5
        BEGIN
                ROLLBACK TRANSACTION
        END
END
```

Триггер к таблице Sales.Currency будет выполняться после вставки, проверяет значение вставляемое в столбце Name и убеждается, что имя составляет менее 5 символов. Если это правило нарушается, то откат транзакции, которая удаляет вновь вставленные строки.

Работа триггера

При работе триггера учитываются следующие условия и ограничения:

- Триггеры могут выполнять откат транзакции, когда не выполняются бизнес-правила
- Когда триггер выполняет откат транзакции отменяется весь пакет
- Любые инструкции ROLLBACK TRANSACTION будут выполнены
- Любые изменения, которые происходят после отката, не откатываются
- Триггер может применяться только к одной таблице
- Триггеры выполняются только в текущей базе данных
- Триггеры должны принадлежать к той же схеме, что и целевая таблица
- Триггеры INSTEAD OF DELETE / UPDATE не могут быть созданы для таблицы, которая имеет каскадное определение внешнего ключа.

Типы триггеров

AFTER

AFTER указывает, что DML триггер срабатывает только тогда, когда все операции, указанные в SQL выражении, были успешно выполнены. Все ссылочные действия и проверки ограничений также должны быть выполнены.

INSTEAD OF

Указывает, что триггер DML выполняется вместо триггера SQL. INSTEAD OF не может быть указан для DDL или триггеров входа.

8.5. Распределенные запросы

Распределенные запросы используются для доступа к данным из нескольких разнородных источников данных. Эти источники данных могут храниться на одном или различных компьютерах. MicrosoftSQL Server поддерживает распределенные запросы с использованием OLE DB.

Пользователи SQL Server могут применять распределенные запросы для доступа к следующим данным:

- Распределенные данные, хранящиеся в нескольких экземплярах SQL Server.
- Разнородные данные, хранящиеся в различных реляционных и нереляционных источниках данных, доступ к которым осуществляется с использованием поставщика OLE DB.

Поставщики OLE DB представляют данные в табличных объектах, именуемых «наборами строк». SQL Server позволяет ссылаться в инструкциях Transact-SQL на наборы строк из поставщиков OLE DB так, как если бы эти наборы строк являлись таблицами SQL Server.

На таблицы и представления внешних источников данных можно ссылаться непосредственно в инструкциях Transact-SQL SELECT, INSERT,

UPDATE и DELETE. Поскольку в распределенных запросах в качестве базового интерфейса используется OLE DB, распределенные запросы могут обращаться к традиционным реляционным СУБД, которые располагают обработчиками SQL-запросов, а также могут обращаться к данным, управляемым источниками данных различной функциональности и сложности. Если программы представляют принадлежащие им данные в табличных наборах строк через поставщика OLE DB, эти данные могут быть использованы в распределенных запросах.

Добавление связанных серверов

Связанный сервер обеспечивает доступ распределенных гетерогенных запросов к источникам данных OLE DB. Связь со связанным сервером создается с помощью sp_addlinkedserver, распределенные запросы могут быть запущены на этом сервере. Если связанный сервер определяется как экземпляр SQL Server, могут быть выполнены удаленные хранимых процедуры.

Когда пользователь входит на локальный сервер и выполняет распределенный запрос, который обращается к таблице на связанном сервере, локальному серверу необходимо войти на связанный сервер от имени пользователя для доступа к этой таблице. Используйте sp_addlinkedsrvlogin чтобы указать учетные данные, которые локальный сервер использует для входа на связанный сервер.

Распределенные запросы могут разрешить пользователям доступ к другим источникам данных (например, к файлам не реляционных источниках данных, таких как Active ™ Directory, и так далее), используя контекст безопасности Microsoft Windows ® учетную запись, под которой работает служба ® SQL Server.

Использование распределенных запросов

Вы можете получить доступ к удаленным разнородным данным, написав специальные распределенные запросы.

Этот метод лучше всего использовать, если удаленные данные доступны не часто.

С помощью специальных распределенных запросов, вы можете избежать создания постоянной ссылки на внешний источник данных и таким образом поддерживать более высокий уровень безопасности.

SQL Sever предоставляет два набора строк функций, таких как OPENROWSET и OPENDATASOURCE, которые могут быть использованы для написания специальных распределенных запросов.

OPENROWSET - содержит все необходимые сведения о соединении, которые требуются для доступа к удаленным данным источника данных OLE DB. Это альтернативный метод для доступа к таблицам на связанном сервере и является однократным нерегламентированным методом соединения и удаленного

доступа к данным с помощью OLE DB. Вместо этого для более частых ссылок на источники данных OLE DB используйте связанные серверы. Функция OPENROWSET может быть использована в предложении FROM запроса так, как если бы она была именем таблицы. Функция OPENROWSET также может быть использована как целевая таблица в инструкциях INSERT, UPDATE или DELETE. Это зависит от возможностей поставщика OLE DB. Несмотря на то, что запрос может возвратить несколько результирующих наборов, функция OPENROWSET возвращает только первый из них.

Функция OPENROWSET также поддерживает массовые операции с помощью встроенного поставщика BULK, позволяющего считывать данные из файла и возвращать их в виде набора строк.

OPENDATASOURCE функция для создания нерегламентированных распределенных запросов. OPENDATASOURCE также может быть использована в качестве первой части из четырех частей имени, которое ссылается на имя таблицы или представления в выражении SELECT, INSERT, UPDATE или DELETE. Она также может быть использована для обозначения удаленных хранимых процедур в EXECUTE заявление. При запуске удаленных хранимых процедур, OPENDATASOURCE должна ссылаться на другой экземпляр SQL Server. OPENDATASOURCE не принимает переменные в качестве своих аргументов.

Написание распределенного запроса к связанному серверу

Вы можете написать запрос к связанному серверу на основе распределенных запросов, используя полное имя из четырех частей и функцию OPENQUERY. Полное имя из четырех частей используется для обозначения данных объектов в связанном сервере.

9. Использование передовых технологий

9.1.План выполнения

Ядро SQL Server Database Engine может показывать, каким образом оно переходит к таблицам и использует индексы для доступа к данным или их обработки для запроса или другой инструкции DM, например для обновления. Это называется выводом плана выполнения. Для проведения анализа медленно выполняемого запроса полезно изучить план выполнения запроса, чтобы определить причину проблемы(Рис.5.).

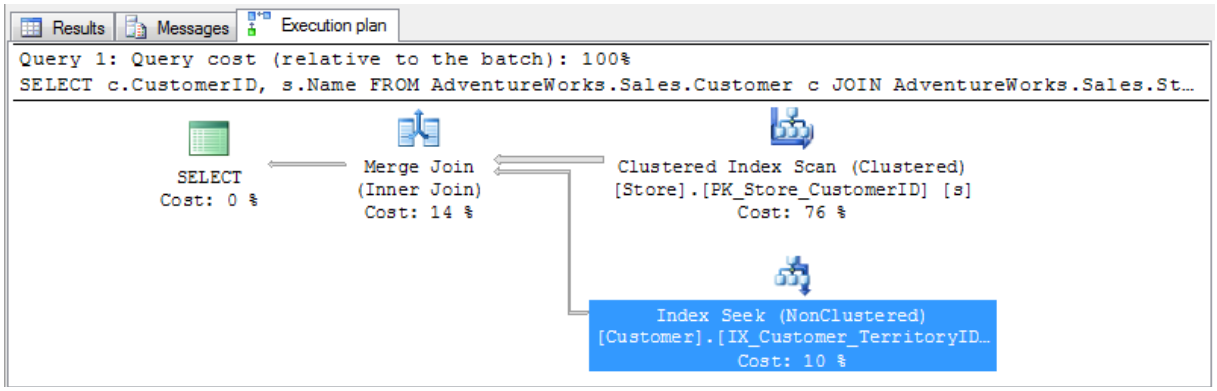


Рис.5. План выполнения запроса.

Планы выполнения отображаются следующими способами.

- SQL Server Management Studio

Отображает либо ориентировочный графический план выполнения (инструкции не выполнены), либо реальный графический план выполнения (при выполненных инструкциях), который можно просмотреть в Management Studio и сохранить.

- Параметры инструкции SET Transact-SQL

При использовании параметров инструкции SET Transact-SQL можно вывести ожидаемый или реальный план выполнения в формате XML или в текстовом формате.

- Классы событий Приложение SQL Server Profiler

Можно включить классы событий Приложение SQL Server Profiler в трассировки для получения ожидаемых или реальных планов выполнения в формате XML или в текстовом формате в результатах трассировки.

При использовании одного из этих способов отображения планов выполнения отображается наилучший план выполнения, используемый ядром Database Engine для отдельных инструкций языка DML и Transact-SQL. В этом плане содержатся сведения о процессе компиляции хранимых процедур и о вызовах хранимых процедур произвольной глубины вложенности. Например, при выполнении инструкции SELECT можно увидеть, что Database Engine выполняет просмотр таблицы для получения данных. Выполнение инструкции SELECT может также показать, что просмотр индекса будет использоваться, если Database Engine определит, что просмотр индекса является наиболее быстрым способом получения данных из таблицы.

9.2.Преобразование типов данных

Преобразование типов данных происходит в следующих случаях:

- При перемещении, сравнении или объединении данных одного объекта с данными другого объекта эти данные могут преобразовываться из одного типа в другой.
- При перемещении в переменную программы данных из результирующего столбца Transact-SQL, кодов возврата или выходных параметров эти

данные должны преобразовываться из системного типа данных SQL Server в тип данных переменной.

Преобразование типов данных бывает явным и неявным.

- Неявное преобразование скрыто от пользователя.

SQL Server автоматически преобразует данные из одного типа в другой. Например, если тип данных **smallint** сравнивается с типом **int**, то перед сравнением тип **smallint** неявно преобразуется в тип **int**.

- Явное преобразование выполняется с помощью функций CAST и CONVERT.

Функции CAST и CONVERT преобразуют значение (локальную переменную, столбец или выражение) из одного типа данных в другой. Например, приведенная ниже функция CAST преобразует числовое значение $157.27 в строку символов '157.27':

При взаимных преобразованиях переменных приложения и столбцов результирующих наборов, кодов возврата, параметров и маркеров параметров SQL Server поддерживаемые преобразования типов данных определяются API базы данных.

Неявные преобразования

Пример:

```
DECLARE @firstname char(10)
SET @firstname =  'Kevin'
```

```
SELECT FirstName, LastName FROM
 Person.Person WHERE
 @firstname = FirstName
```

В базе данных AdventureWorks2008, поле FirstName в таблице Person.Person типа данных NVARCHAR, в примере кода требуется неявное преобразование, чтобы сравнить его и значение переменной @ firsname, которая является символьным типом данных.

Неявное преобразование показано в плане выполнения запроса (Рис.6.).

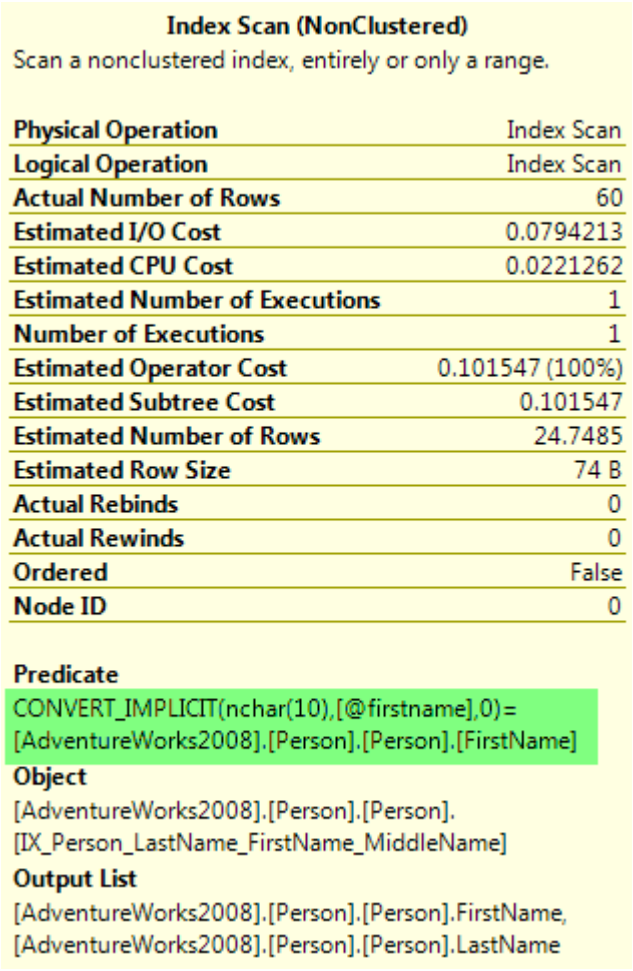


Рис.6. Неявное преобразование в плане выполнения запроса.

Нет неявного преобразования для следующих типов данных: текст и изображения, sql_variant, XML, CLR User-Defined, и HierarchyID.

Явные преобразования

Примеры:

```
USE AdventureWorks2008;
GO
SELECT SUBSTRING(Name, 1, 30) AS ProductName, ListPrice
FROM Production.Product
WHERE CAST(ListPrice AS int) LIKE '3%';
GO
```

```
USE AdventureWorks2008;
GO
SELECT SUBSTRING(Name, 1, 30) AS ProductName, ListPrice
FROM Production.Product
WHERE CONVERT(int, ListPrice) LIKE '3%';
GO
```

CAST и CONVERT выполняют практически идентичные задачи преобразования типов данных.

CAST совместим со стандартом SQL-92, и, следовательно, более компактен.

CONVERT является специфическим для SQL Server и предоставляет большую гибкость, чем CAST при преобразовании типов данных даты и времени.

Приоритеты при преобразовании типов данных

Если оператор связывает два выражения различных типов данных, то по правилам приоритета типов данных определяется, какой тип данных имеет меньший приоритет и будет преобразован в тип данных с большим приоритетом. Если неявное преобразование не поддерживается, возвращается ошибка. Если оба операнда выражения имеют одинаковый тип данных, результат операции будет иметь тот же тип данных.

В SQL Server используется следующий приоритет типов данных:

1. Определяемый пользователем типы данных (высший приоритет);
2. sql_varian t;
3. xml;
4. datetimeoffset;
5. datetime2;
6. datetime;
7. smalldatetime;
8. date;
9. time;
10. float;
11. real;
12. decimal;
13. money;
14. smallmoney;
15. bigint;
16. int;
17. smallint;
18. tinyint;
19. bit;
20. ntext;
21. text;
22. image;
23. timestamp;
24. uniqueidentifier;
25. nvarchar (включая nvarchar(max));

26. nchar;
27. varchar (включая varchar(max));
28. char;
29. varbinary (включая varbinary(max));
30. binary (низший приоритет).

Примеры:
Без явного преобразования не выполняется:

```
DECLARE @label varchar(12),
        @pageno int
SET @label='Page Number '
SET @pageno = 1
Print @label + @pageno
```

С явным преобразованием выполнится успешно.

```
DECLARE @label varchar(12),
        @pageno int
SET @label='Page Number '
SET @pageno = 1
Print @label + CONVERT(varchar, @pageno)
```

9.3. Работа с типами данных

Использование типов данных типа дата время

Для всех типов данных даты и времени поддерживаются реляционные операторы (<, <=, >, >=, <>), операторы сравнения (=, <, <=, >, >=, <>, !<, !>), а также логические операторы и предикаты Boolean (IS NULL, IS NOT NULL, IN, BETWEEN, EXISTS, NOT EXISTS и LIKE).

Арифметические операторы для типов данных даты и времени

Для сложения и вычитания любых типов данных даты и времени следует использовать функции DATEADD и DATEDIFF.

Использование форматов даты и времени

Форматы строковых литералов влияют на представление данных в приложениях для пользователей, но не на базовый формат хранения целых чисел в SQL Server. Однако SQL Server может воспринять значение даты в формате строкового литерала, поставленное приложением или пользователем для хранения или для обработки функцией для работы с датами, как разные даты. То, как будет воспринято значение, зависит от сочетания формата строкового лите-

рала, типа данных и следующих настроек времени выполнения: SET DATEFORMAT, SET LANGUAGE и параметр default language.

На некоторые форматы строковых литералов эти настройки не влияют. Если нет уверенности в том, что эти настройки верны для данного формата, то, возможно, стоит задуматься об использовании формата, не зависящего от них. Формат ISO 8601 не зависит от этих настроек и является международным стандартом. Язык Transact-SQL, который использует форматы строковых литералов, зависящие от системных настроек, является менее мобильным.

Формат даты **ydm** не поддерживается для типов **date**, **datetime2**.и **datetimeoffset**. Во время выполнения возникнет ошибка.

Построение иерархического типа данных

Тип данных hierarchyid является системным типом данных переменной длины. Тип данных hierarchyid используется для представления положения в иерархии. Столбец типа hierarchyid не принимает древовидную структуру автоматически. Приложение должно создать и назначить значения hierarchyid таким образом, чтобы они отражали требуемые связи между строками.

Значение типа данных hierarchyid представляет позицию в древовидной иерархии. Значения hierarchyid обладают следующими свойствами.

- Исключительная компактность

Среднее число бит, необходимое для представления узла в древовидной структуре с *n* узлами, зависит от среднего количества потомков у узла. Для небольших уровней ветвления (0 - 7) этот размер равен 6*logA*n* бит, где A — средний уровень ветвления. Для представления узла в иерархии организации, насчитывающей 100 000 человек со средним уровнем ветвления 6, необходимо около 38 бит. Эта величина округляется до 40 бит (5 байт), которые необходимы для хранения.

- Сравнение проводится в порядке приоритета глубины

Если заданы два значения hierarchyid**a** и **b**, **a<b** означает, что значение a появляется раньше значения b, если проходить по дереву с приоритетным направлением в глубину. Индексы для типов данных hierarchyid располагаются в порядке приоритета глубины, и узлы, встречающиеся рядом при проходе по дереву с приоритетным направлением глубины, хранятся рядом друг с другом. Например, потомки некоторой записи хранятся рядом с этой записью.

- Поддержка произвольных вставок и удалений

С помощью метода GetDescendant можно в любой момент создать одноуровневый элемент, расположенный справа от заданного узла, слева от заданного узла или между любыми двумя другими одноуровневыми элементами. Свойство сравнения сохраняется, если произвольное число узлов вставляется в иерархию или удаляется из нее. Большинство операций вставки и удаления сохраняют свойство компактности. Однако операции вставки между двумя узлами приводят к созданию значений hierarchyid, обладающих менее компактным представлением.

- Кодировка в типе данных hierarchyid ограничена 892 байтами. Следовательно, узлы, имеющие слишком много уровней, чтобы уместиться в 892 байта, не могут быть представлены типом данных hierarchyid.

Тип данных hierarchyid доступен клиентам среды CLR в виде типа данных SqlHierarchyId.

Тип данных hierarchyid логически кодирует сведения об одном узле в дереве иерархии, кодируя путь от корня дерева к этому узлу. Такой путь логически представлен в виде последовательности меток всех посещенных дочерних узлов, начиная с корня. Представление начинается косой чертой, а путь к корню представлен одной косой чертой. Для уровней ниже корня каждая метка кодируется в виде последовательности целых чисел, разделенных точками. Сравнения дочерних узлов выполняется путем сравнения этих целочисленных последовательностей, разделенных точками, в лексикографическом порядке. Уровни разделяются косой чертой. То есть косая черта отделяет родительский узел от дочернего. Например, следующие пути с длиной 1, 2, 2, 3 и 3 уровня соответственно действительны для типа hierarchyid.

- /
- /1/
- /0.3.-7/
- /1/3/
- /0.1/0.2/

Узлы можно вставлять в любое место. Узлы, вставленные после **/1/2/**, но перед **/1/3/**, можно представить как **/1/2.5/**. Узлы, вставленные после 0, логически представлены в виде отрицательных чисел. Например, узел, расположенный перед узлом **/1/1/**, можно представить как **/1/-1/**. Узлы не должны начинаться с нулей. Например, узел **/1/1.1/** является допустимым, а узел **/1/1.01/** — недопустим. Чтобы избежать ошибок, вставляйте узлы с помощью метода GetDescendant.

Работа с иерархией

Существуют два подхода к индексированию иерархических данных:

1.В глубину (Рис.7.)

Если используется индекс преимущественно в глубину, строки поддерева хранятся рядом друг с другом. Например, записи всех сотрудников, в цепи подчиненности которых есть данный руководитель, хранятся рядом с записью руководителя.

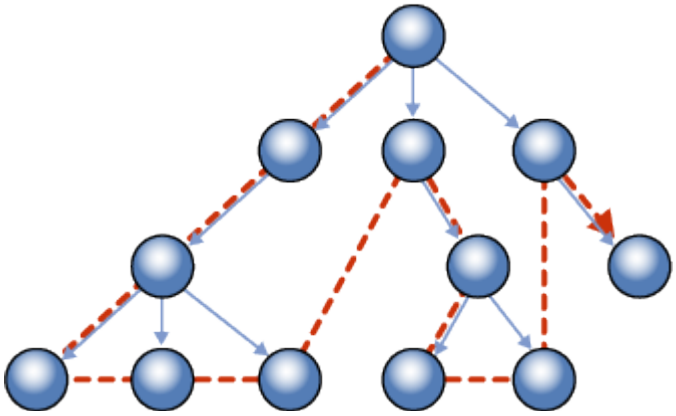


Рис.7. Индекс в глубину.

Пример:

```
CREATE TABLE Organization
  (
   EmployeeID hierarchyid,
   OrgLevel as EmployeeID.GetLevel(),
   EmployeeName nvarchar(50) NOT NULL
  ) ;
GO
```

Если используется индексирование в ширину, строки одного уровня иерархии хранятся вместе. Например, записи всех сотрудников, напрямую подчиненных одному и тому же руководителю, хранятся рядом друг с другом.

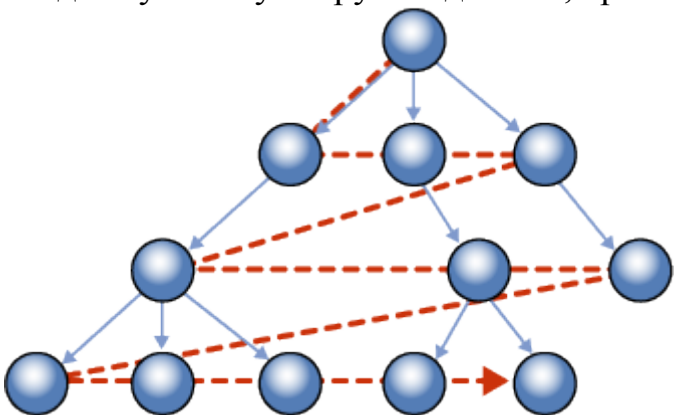


Рис.8. Индекс в ширину.

Пример:

```
CREATE UNIQUE INDEX Org_Depth_First
ON Organization(EmployeeID);
GO
```

9.4. Курсоры и Set-Base запросы

Операции в реляционной базе данных выполняются над множеством строк. Набор строк, возвращаемый инструкцией SELECT, содержит все строки, которые удовлетворяют условиям, указанным в предложении WHERE инструкции. Такой полный набор строк, возвращаемых инструкцией, называется ре-

зультирующим набором. Приложения, особенно интерактивные, не всегда эффективно работают с результирующим набором как с единым целым. Им нужен механизм, позволяющий обрабатывать одну строку или небольшое их число за один раз.

Курсоры являются расширением результирующих наборов, которые предоставляют такой механизм.

Курсоры позволяют усовершенствовать обработку результатов:

- позиционируясь на отдельные строки результирующего набора;
- получая одну или несколько строк от текущей позиции в результирующем наборе;
- поддерживая изменение данных в строках в текущей позиции результирующего набора;
- поддерживая разные уровни видимости изменений, сделанных другими пользователями для данных, представленных в результирующем наборе;
- предоставляя инструкциям Transact-SQL в сценариях, хранимых процедурах и триггерах доступ к данным результирующего набора.

Реализация курсоров в Transact-SQL

Курсоры основываются на синтаксисе DECLARE CURSOR и используются главным образом в сценариях Transact-SQL, хранимых процедурах и триггерах. Курсоры Transact-SQL реализуются на сервере и управляются инструкциями Transact-SQL, направляемыми клиентом серверу. Они также могут содержаться в пакетах, хранимых процедурах или триггерах.

Серверные курсоры интерфейса прикладного программирования (API) - поддерживают функции курсоров API в OLE DB и ODBC. Курсоры API реализуются на сервере. Всякий раз, когда клиентское приложение вызывает функцию курсора API, поставщик OLE DB или драйвер ODBC для собственного клиента SQL Server передает требование на сервер для выполнения действия в отношении серверного курсора API.

Клиентские курсоры - реализуются внутренне драйвером ODBC для собственного клиента SQL Server и DLL, реализующим API-интерфейс ADO. Клиентские курсоры реализуются посредством кэширования всех строк результирующего набора на клиенте. Каждый раз, когда клиентское приложение вызывает функцию курсора API, драйвер ODBC для собственного клиента SQL Server или ADO DLL выполняет операцию курсора на строках результирующего набора, кэшированных на клиенте.

Поскольку курсоры Transact-SQL и серверные курсоры API реализуются на сервере, они собирательно называются серверными курсорами.

Не следует смешивать использование этих различных типов курсоров. При выполнении инструкции DECLARE CURSOR с ключевым словом OPEN из приложения сначала необходимо присвоить атрибутам курсора API их значения по умолчанию. Если присвоить атрибутам курсора API значения, отличные от их значений по умолчанию, а затем выполнить инструкцию DECLARE

CURSOR с ключевым словом OPEN, это рассматривается как требование к SQL Server установить соответствие курсора API и курсора Transact-SQL. Например, нельзя определить значения атрибутов ODBC, требующих сопоставления курсора, управляемого набором ключей, с результирующим набором, а затем использовать эту инструкцию для выполнения вызова инструкции DECLARE CURSOR с ключевым словом OPEN для курсора INSENSITIVE.

Возможным недостатком серверных курсоров является то, что они в настоящее время поддерживают не все инструкции Transact-SQL. Серверные курсоры не поддерживают инструкции Transact-SQL, создающие несколько результирующих наборов; следовательно, они не могут быть использованы при выполнении приложением хранимой процедуры или пакета, содержащих более одной инструкции SELECT. Серверные курсоры также не поддерживают инструкции SQL, содержащие ключевые слова COMPUTE, COMPUTE BY, FOR BROWSE или INTO.

Использование курсоров.

Для реализации курсоров используется следующая последовательность:

- Связать и определить характеристики
- Заполнить курсор
- Получить строки в курсоре
- Изменить данные в случае необходимости
- Закрыть и освободить курсор

Пример:

```
DECLARE vend_cursor CURSOR
  FOR SELECT * FROM Purchasing.Vendor
OPEN vend_cursor
FETCH NEXT FROM vend_cursor
CLOSE vend_cursor
DEALLOCATE vend_cursor
```

DECLARE CURSOR определяет такие атрибуты серверного курсора языка Transact-SQL, как свойства просмотра и запрос, используемый для построения результирующего набора, на котором работает курсор. Инструкция DECLARE CURSOR поддерживает как синтаксис стандарта ISO, так и синтаксис, использующий набор расширений языка Transact-SQL.

DEALLOCATE удаляет ссылку курсора. Когда удаляется последняя ссылка курсора, SQL Server освобождает структуры данных, составляющие курсор.

Логика Set-Based

Логика Set-Based:

- SQL Server перебирает данные
- Результат возвращается в виде набора, а не строка за

Строкой.

Пример:

```
SELECT ProductID, Purchasing.Vendor.VendorID, Name
FROM Purchasing.ProductVendor JOIN Purchasing.Vendor
   ON (Purchasing.ProductVendor.VendorID = Purchasing.Vendor.VendorID)
WHERE StandardPrice > $10
  AND Name LIKE N'F%'
GO
```

С помощью соединения можно получать данные из двух или нескольких таблиц на основе логических связей между ними. Соединения указывают, как MicrosoftSQL Server должен использовать данные из одной таблицы для выбора строк из другой таблицы.

Соединение определяет способ связывания двух таблиц в запросе следующим образом:

- для каждой таблицы указываются столбцы, используемые в соединении. В типичном условии соединения указывается внешний ключ из одной таблицы и связанный с ним ключ из другой таблицы;
- указывается логический оператор (например, = или <>,) для сравнения значений столбцов.

Внутреннее соединение можно задавать в предложениях FROM и WHERE. Внешнее соединение можно указывать только в предложении FROM. Условия соединения сочетаются с условиями поиска WHERE и HAVING для управления строками, выбранными из базовых таблиц, на которые ссылается предложение FROM.

9.5. Динамический SQL

Хотя статический SQL хорошо работает во многих ситуациях, существует класс приложений, в которых доступ к данным не может быть определен заранее. Например, предположим, что таблица позволяет пользователям вводить запрос, который затем отправляет таблицу к базе данных для получения данных. Содержание этого запроса, очевидно, не может быть известна программисту, когда пишется программа.

Чтобы решить эту проблему, используется встроенный SQL называется динамическим SQL. В отличие от статического SQL, когда столбцы жестко за-

кодированы в программе, динамические операторы SQL могут быть построены во время выполнения и помещается в переменную строку хоста. Затем они направляются в СУБД для обработки. Потому что СУБД должен генерировать план доступа во время выполнения для динамических операторов SQL, выполнение динамического SQL, как правило, медленнее, для статического SQL.

Пример:

```
SET @SQLString = N'SELECT @SalesOrderOUT = MAX(SalesOrderNumber)
   FROM Sales.SalesOrderHeader
   WHERE CustomerID = @CustomerID';
```

Использование sp_executesql. sp_executesql является предпочтительным методом для выполнения динамических операторов SQL, поскольку он является более универсальным и эффективным, чем EXECUTE

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

Использование EXECUTE

```
[ { EXEC | EXECUTE } ]
  { [ @return_status = ]
    { module_name [ ;number ] | @module_name_var }
     [ [ @parameter = ] { value  | @variable [ OUTPUT ]
       | [ DEFAULT ] } ] [ ,...n ] [ WITH RECOMPILE ] } [;]
```

С динамическим SQL, план запроса, как правило, не кэшируется и SQL Server должен перестроить план запроса во время выполнения. Использование sp_executesql дает наилучший шанс для SQL Server, повторно кэше план запроса; EXECUTE не сохраняет планы запросов.

Динамический SQL обеспечивает открытие для атак SQL Injection. Атака путем внедрения кода SQL — это атака, при которой производится вставка вредоносного кода в строки, передающиеся затем в экземпляр SQL Server для синтаксического анализа и выполнения. Любая процедура, создающая инструкции SQL, должна рассматриваться на предмет уязвимости к внедрению небезопасного кода, поскольку SQL Server выполняет все получаемые синтаксически правильные запросы. Даже параметризованные данные могут стать предметом манипуляций опытного злоумышленника.

Основная форма атаки путем внедрения кода SQL состоит в прямой вставке кода в пользовательские входные переменные, которые объединяются с командами SQL и выполняются. Менее явная атака внедряет небезопасный код в строки, предназначенные для хранения в таблице или в виде метаданных. Когда впоследствии сохраненные строки объединяются с динамической командой SQL, происходит выполнение небезопасного кода.

Атака осуществляется посредством преждевременного завершения текстовой строки и присоединения к ней новой команды. Поскольку к вставленной команде перед выполнением могут быть добавлены дополнительные строки, злоумышленник заканчивает внедряемую строку меткой комментария «--». Весь последующий текст во время выполнения не учитывается.

9.6. Управление файлами запросов

Управление версиями

Не стоит забывать о необходимости контроля версий для управления SQL Server проекты, которые могут иметь много разработчиков. Контроль версий позволяет вносить изменения, которые отслеживаются, и отслеживает изменения файлов, так что любые текущие или ранние версии могут быть восстановлены.

Team Foundation Server

Сервер Team Foundation Server поддерживает только SQL Server 2005 в качестве базы данных учетных до RTM из Team Foundation Server SP1 и SQL Server 2008.

TFS может выполнять контроль версий проектов для SQL Server 2008. Visual SourceSafe 2005 также может быть использован для управления версиями.

Team Foundation Server представляет собой интегрированный сервер совместной работы для визуальных Studio Team System. Он сочетает в себе контроль версий, отслеживание рабочих элементов, управление сборками, управление процессами и бизнес-аналитики на едином сервере.

Это позволяет всем членам группы сотрудничать более эффективно и повышать качество программного обеспечения.

Литература

1. Электронная документация по Microsoft SQL Server http://msdn.microsoft.com/ru-ru/library/bb418440(v=SQL.10).aspx.
2. MSDN Library. SQL Server 2008 http://msdn.microsoft.com/ru-ru/library/bb418491.aspx.
3. Database Journal. The Knowledge Center for Database Professionals. Электронный журнал. http://www.databasejournal.com/features/article.php/3593466/MS-SQL-Series.htm.
4. Крис Дж. Дейт. SQL и реляционная теория. Как грамотно писать код на SQL (SQL and Relational Theory: How to Write Accurate SQL Code), Символ-Плюс, 2010.
5. Линн Бейли. Изучаем SQL (Head First SQL), Питер, 2012.
6. Эйри Джоунс, Райан К. Стивенз, Рональд Р. Плю, Роберт Ф. Гарретт, Алекс Кригель. Функции SQL. Справочник программиста. SQL Functions: Programmer's Reference, Вильямс, 2006.
7. Джо Селко. SQL для профессионалов. Программирование SQL for Smarties: Advanced SQL Programming, Лори, 2009.
8. Владислав Пирогов. SQL Server 2005. Программирование клиент-серверных приложений. (+CD-ROM), БХВ - Петербург, 2006.
9. Ицик Бен-Ган. Microsoft SQL Server 2008. Основы T-SQL Microsoft SQL Server 2008: T-SQL Fundamentals, БХВ - Петербург, Русская Редакция, 2009.
10. Брайан Найт, Кетан Пэтел, Вейн Снайдер, Росс Лофорт, Стивен Уорт. Microsoft SQL Server 2008. Руководство администратора для профессионалов Professional Microsoft SQL Server 2008 Administration, Диалектика, Вильямс, 2010.
11. Робин Дьюсон. SQL Server 2008 для начинающих разработчиков Beginning SQL Server 2008 for Developers From Novice to Professional, БХВ – Пертербург, 2009.
12. Майк Гандерлой, Джозеф Джорден, Дейвид Чанц. Освоение
13. Microsoft SQL Server Analysis Services 2008 и MDX для профессионалов Microsoft SQL Server Analysis Services 2008 with MDX, Вильямс, 2010.
14. Под редакцией Александра Гладченко и Владислава Щербинина. Репликация Microsoft SQL Server 2005/2008,Эком Паблишерз, 2009.
15. Душан Петкович. Microsoft SQL Server 2008. Руководство для начинающих Microsoft SQL Server 2008: A Beginner's Guide, БХВ – Петербург, 2009.
16. Джеми Макленнен, Чжаохуэй Танг, Богдан Криват. Microsoft SQL Server 2008. Data Mining - интеллектуальный анализ данных Data Mining with Microsoft SQL Server 2008, БХВ – Петербург, 2009.

17. Тобиаш Тернстрем, Энн Вебер, Майк Хотек. Microsoft SQL Server 2008. Разработка баз данных. Учебный курс Microsoft (+ CD-ROM) MCTS Self-Paced Training Kit (Exam 70-433): Microsoft SQL Server 2008-Database Development. Русская Редакция, 2010.
18. Роберт Виейра. Программирование баз данных Microsoft SQL Server 2008. Базовый курс. Beginning Microsoft SQL Server 2008 Programming, Диалектика, Вильямс, 2010.
19. Роберт Уолтерс, Майкл Коулс, Фабио Клаудио Феррачати, Роберт Рей, Дональд Фармер.SQL Server 2008. Ускоренный курс для профессионалов Accelerated SQL Server 2008, Вильямс, 2009.
20. Уильям Р. Станек. Microsoft SQL Server 2008. Справочник администратора. Microsoft SQL Server 2008: Administrator's Pocket Consultant, БХВ – Петербург, Русская Редакция, 2009.
21. Леонард Лобел, Эндрю Дж. Браст, Стивен Форте. Разработка приложений на основе Microsoft SQL Server 2008. Programming Microsoft SQL Server 2008, БХВ – Петербург, Русская Редакция, 2010.
22. Майк Хотек. Microsoft SQL Server 2008. Реализация и обслуживание. Учебный курс Microsoft (+ CD-ROM). MCTS Self-Paced Training Kit (Exam 70-432): Microsoft SQL Server 2008 Implementation and Maintenance, Русская Редакция, 2011.
23. #2778 Writing Queries Using Microsoft® SQL Server™ 2008 Transact-SQL. Официальный курс Microsoft.

В 2009 году Университет стал победителем многоэтапного конкурса, в результате которого определены 12 ведущих университетов России, которым присвоена категория «Национальный исследовательский университет». Министерством образования и науки Российской Федерации была утверждена программа его развития на 2009–2018 годы. В 2011 году Университет получил наименование «Санкт-Петербургский национальный исследовательский университет информационных технологий, механики и оптики»

Кафедра Программных систем

Кафедра **Программных систем** входит в состав нового факультета **Инфокоммуникационные технологии**, созданного решением Ученого совета университета 17 декабря 2010 г. по предложению инициативной группы сотрудников, имеющих большой опыт в реализации инфокоммуникационных проектов федерального и регионального значения.

На кафедре ведется подготовка бакалавров и магистров по направлению **210700 «Инфокоммуникационные технологии и системы связи»**:

210700.62.10 – ИНТЕЛЛЕКТУАЛЬНЫЕ ИНФОКОММУНИКАЦИОННЫЕ СИСТЕМЫ (Бакалавр)

210700.68.05 – ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ В ИНФОКОММУНИКАЦИЯХ (Магистр)

Выпускники кафедры получают фундаментальную подготовку по: математике, физике, электронике, моделированию и проектированию инфокоммуникационных систем (ИКС), информатике и программированию, теории связи и теории информации.

В рамках профессионального цикла изучаются дисциплины: архитектура ИКС, технологии программирования, ИКС в Интернете, сетевые технологии, администрирование сетей Windows и UNIX, создание программного обеспечения ИКС, Web программирование, создание клиент-серверных приложений.

Область профессиональной деятельности бакалавров и магистров включает:

- сервисно-эксплуационная в сфере современных ИКС;
- расчетно-проектная при создании и поддержке сетевых услуг и сервисов;

- экспериментально-исследовательская;
- организационно-управленческая – в сфере информационного менеджмента ИКС.

Знания выпускников востребованы:
- в технических и программных системах;
- в системах и устройствах звукового вещания, электроакустики, речевой, и мультимедийной информатики;
- в средствах и методах защиты информации;
- в методах проектирования и моделирования сложных систем;
- в вопросах передачи и распределения информации в телекоммуникационных системах и сетях;
- в методах управления телекоммуникационными сетями и системами;
- в вопросах создания программного обеспечения ИКС.

Выпускники кафедры Программных систем обладают компетенциями:
- проектировщика и разработчика структур ИКС;
- специалиста по моделированию процессов сложных систем;
- разработчика алгоритмов решения задач ИКС;
- специалиста по безопасности жизнедеятельности ИКС;
- разработчика сетевых услуг и сервисов в ИКС;
- администратора сетей: UNIX и Windows;
- разработчика клиентских и клиент-серверных приложений;
- разработчика Web – приложений;
- специалиста по информационному менеджменту;
- менеджера проектов планирования развития ИКС.

Трудоустройство выпускников:
1. ОАО «Петербургская телефонная сеть»;
2. АО «ЛЕНГИПРОТРАНС»;
3. Акционерный коммерческий Сберегательный банк Российской Федерации;
4. ОАО «РИВЦ-Пулково»;
5. СПБ ГУП «Петербургский метрополитен»;
6. ООО «СоюзБалтКомплект»;
7. ООО «ОТИС Лифт»;
8. ОАО «Новые Информационные Технологии в Авиации»;
9. ООО «Т-Системс СиАйЭс» и др.

Кафедра сегодня имеет в своем составе высококвалифицированный преподавательский состав, в том числе:
- 5 кандидатов технических наук, имеющих ученые звания профессора и доцента;

- 4 старших преподавателя;
- 6 штатных совместителей, в том числе кандидатов наук, профессиональных IT - специалистов;
- 15 Сертифицированных тренеров, имеющих Западные Сертификаты фирм: Microsoft, Oracle, Cisco, Novell.

Современная техническая база; лицензионное программное обеспечение; специализированные лаборатории, оснащенные необходимым оборудованием и ПО; качественная методическая поддержка образовательных программ; широкие Партнерские связи существенно влияют на конкурентные преимущества подготовки специалистов.

Авторитет специализаций кафедры в области компьютерных технологий подтверждается Сертификатами на право проведения обучения по методикам ведущих Западных фирм - поставщиков аппаратного и программного обеспечения.

Заслуженной популярностью пользуются специализации кафедры ПС по подготовке и переподготовке профессиональных компьютерных специалистов с выдачей Диплома о профессиональной переподготовке по направлениям: "Информационные технологии (инженер-программист)" и "Системный инженер", а также Диплома о дополнительном (к высшему) образованию с присвоением квалификации: "Разработчик профессионально-ориентированных компьютерных технологий ". В рамках этих специализаций высокопрофессиональные преподаватели готовят компетентных компьютерных специалистов по современным в России и за рубежом операционным системам, базам данных и языкам программирования ведущих фирм: Microsoft, Cisco, IBM, Intel, Oracle, Novell и др.

Профессионализм, компетентность, опыт, и качество программ подготовки и переподготовки IT- специалистов на кафедре ПС неоднократно были удостоены высокими наградами «Компьютерная Элита» в номинации лучший учебный центр России.

Партнеры:

1. **Microsoft** Certified Learning Solutions;
2. **Novell** Authorized Education Center;
3. **Cisco** Networking Academy;
4. **Oracle** Academy;
5. **Sun Java** Academy и др;
6. **Prometric**;
7. **VUE**.

Мы готовим квалифицированных инженеров в области инфокоммуникационных технологий с новыми знаниями, образом мышления и способностями быстрой адаптации к современным условиям труда.

Татьяна Викторовна Зудилова
Галина Юрьевна Шмелева

Создание запросов в Microsoft SQL Server 2008

УЧЕБНОЕ ПОСОБИЕ

В авторской редакции
Редакционно-издательский отдел НИУ ИТМО
Зав. РИО Н.Ф. Гусарова
Лицензия ИД № 00408 от 05.11.99
Подписано к печати
Заказ №
Тираж 100 экз.
Отпечатано на ризографе

Редакционно-издательский отдел
Санкт-Петербургского национального
исследовательского университета
информационных технологий, механики
и оптики

197101, Санкт-Петербург, Кронверкский пр., 49