Репликация SQL Server 2005/2008

Сборник статей от сообщества SQL.RU

Под общей редакцией
Александра Гладченко и Владислава Щербинина

Москва

ЭКОМ

2009

ББК 32.97
УДК 681.3
Р42

Р42 **Репликация SQL Server 2005/2008.** Сборник статей от сообщества SQL.RU. Под общей редакцией А. Гладченко и В. Щербинина. – М.: ЭКОМ Паблишерз, 2008. –288 с.: ил.

ISBN: 978-5-9790-0086-2

Оглавление

Вместо введения 5
Основы репликации SQL Server 2008 5
Общие вопросы 47
Резервное копирование в репликации SQL Server 2008 48
Шпаргалка по репликации в Microsoft SQL Server 57
SQL Server 2005: Графический интерфейс репликации 67
Вы все еще используете предварительные и заключительные сценарии моментального снимка? 72
BCP-секционирование в SQL Server 2005 74
Посредники, учетные данные и подсистемы репликации 79
Использование утилиты TableDiff в репликации 84
Репликация DDL в SQL Server 2000 и 2005 98
Репликация программируемых объектов БД в SQL Server 2005 101
Репликация полнотекстовых индексов в SQL Server 2005 113
Модель репликации с переизданием 117
Репликация транзакций 121
Высокая доступность в Репликации SQL Server 2008 с зеркалированием и Доставкой журналов 122
Настройка транзакционной репликации в SQL Server 2005 148
Обслуживание транзакционной репликации в SQL Server 2005 169
Мониторинг и решение проблем при репликации в SQL Server 2005 180
Агенты репликации транзакций в SQL Server 2005 196
Настройка одноранговой репликации транзакций 205
Одноранговая репликация транзакций в Microsoft SQL Server 2005 210
Управление одноранговой репликацией в SQL Server 2005 218
Изменения в репликации транзакций: трассировочные маркеры 227
Инициализация подписки без первоначальной синхронизации данных в SQL Server 2005 230
Репликация DML 234

Репликация слиянием 247
Пакетные обновления в репликации слиянием в SQL Server 2005 248
Изменения в репликации слиянием: Динамические фильтры и однонаправленная репликация 254
Веб-синхронизация в SQL 2005 для репликации слиянием 260
Зеркальное отображение базы данных и репликация слиянием 268
Репликация слиянием изнутри: как идентифицировать ожидающие синхронизации изменения 271
Усовершенствования очистки метаданных в SQL Server 2005 278

Вместо введения

Основы репликации SQL Server 2008

Автор Александр Юрьевич Гладченко

Репликация – это одна из разновидностей систем, поддерживающих распределенную обработку данных. В последнее десятилетие направление распределенной обработки данных бурно развивалось, и на сегодняшний день это одно из наиболее динамично растущих направлений. В докладе об изменении профиля корпоративных данных, который был озвучен на саммите APAC, посвященном хранилищам данных и состоявшемся в 2007г. в Хошимине, Рик Вилларс (Rick Villars, Head of Investment Technical Services HSBC) показал, что объем тиражируемых данных увеличивается ежегодно на 43.9% и к 2009 г. сравняется с объемом традиционных данных, размещаемых в хранилищах. Назначение этой книги состоит в том, чтобы предоставить читателям набор апробированных в течение нескольких лет рецептов по использованию и настройке репликации в SQL Server.

В этой статье я рассмотрю наиболее значимые для репликации концептуальные моменты, и сделаю небольшой обзор основных направлений администрирования и мониторинга внутренних подсистем репликации SQL Server.

Назначение репликации состоит в управляемом тиражировании данных между несколькими СУБД. Задача эта не нова, и поэтому в репликации используется давно всем известная метафора издательского дела: издатель, публикация, статья, распространитель и подписчик. Для SQL Server характерно также наличие специализированных программных модулей, с помощью которых организуется взаимодействие задействованных в репликации баз данных, и которые в документации принято называть агентами репликации.

Вот определение репликации, которое было представлено в учебном пособии Майкрософт: «Репликация – это процесс автоматического распределения копий данных и объектов БД между экземплярами SQL Server с одновременной синхронизацией всей распространяемой информации». Известный теоретик реляционной теории баз данных Крис Дейт дал репликации такое определение: «Система поддерживает репликацию данных, если данная таблица или ее фрагмент может быть представлена несколькими отдельными копиями или репликами, которые хранятся на нескольких отдельных узлах». Однако, ни одно из приведенных тут определений мне не кажется полным. В профессиональных интернет-форумах очень часто можно встретить вопрос, как организо-

вать репликацию без прямого подключения серверов друг к другу, например, посредством электронной почты? Мне видится неверной даже сама постановка вопроса таким образом. Дело в том, что нужно четко отделять репликацию от простого тиражирования данных. По моему мнению, распределенная система может называться системой с репликацией, если она не только автоматически синхронизирует данные по заданным правилам, но и умеет гарантировать их синхронность. Например, если для доставки изменений в данных используется протокол без какой-либо гарантии доставки, такую систему уже нельзя назвать репликацией, хотя это еще не означает, что данные в такой системе будут не синхронны. С другой стороны, репликация также является очень эффективным способом тиражирования данных, но она не является единственной стратегией тиражирования данных. Давайте кратко рассмотрим несколько стратегий тиражирования данных, которые также доступны на платформе Майкрософт.

В первую очередь стоит упомянуть распределенные транзакции, которые могут передаваться по сети с довольно низкой задержкой. Для успешного функционирования распределенных транзакций необходимо иметь возможность транспортировки их между узлами, координации этих транзакций и все узлы и участники транспортной системы распределенных транзакций в момент прохождения такой транзакции должны быть активны. Разумеется, такая сложная цепочка не гарантирует автономности узлов, и отслеживание передачи изменений на все узлы возлагается на разработчика приложения. Подобные проблемы присущи и распределенным запросам, механизм которых схож с распределенными транзакциями, и отличается только отсутствием компонент координатора распределенных транзакций.

Для копирования данных с одного сервера на другой иногда бывает достаточно простого резервного копирования базы данных с последующим восстановлением из резервной копии на другом сервере. Существует даже методика доставки журналов, которая позволяет средствами резервного копирования и восстановления поддерживать актуальную копию базы данных на другом сервере. Такая технология хорошо подходит для дублирования всей базы данных без возможности внесения изменений в копию, кроме того, имеются известные проблемы с доступностью базы данных во время восстановления из копий. Очень близкой по смыслу к технологии доставки журналов является зеркальное отображение базы данных. В последнем случае дублирование базы данных предназначено для обеспечения горячего или теплого резерва базы данных на случай аппаратных или программных сбоев. Переключение запросов пользователей на зеркальную копию может выполняться автоматически.

Службы интеграции данных в составе SQL Server также предоставляют богатый набор средств тиражирования информации. С помощью пакетов Data Transformation Services (DTS) можно не только импортировать/экспортировать схему и данные между узлами с SQL Server или любых

других поддерживаемых провайдерами источников данных, возможна самая разнообразная трансформация передаваемых между узлами данных. Пакеты DTS могут передаваться также как и резервные копии и даже использоваться для передачи изменений в стандартной репликации.

Бурное развитие мобильных устройств привело к появлению, например, такой новой технологии синхронизации данных как Microsoft Synchronization Services для ADO.NET. Эта новая служба позволяет синхронизировать данные из разных источников, используя двухуровневую или многоуровневую архитектуру взаимодействия специализированных служб. Вместо копирования данных и схемы базы, прикладной программный интерфейс службы синхронизации предоставляет в распоряжение разработчика набор компонентов, с помощью которых можно синхронизировать данные между службами предоставления данных и локальным хранилищем данных пользователя. Такая архитектура ориентирована в первую очередь на тех мобильных клиентов, которые не имеют гарантированного и надежного подключения по сети к центральному серверу, и которые могут работать какое-то время с локальной копией данных автономно. В данном случае причиной тиражирования данных является необходимость временного кэширования информации на устройстве клиента-подписчика.

Перечисленные способы тиражирования информации без репликации предназначены для решения отдельных, зачастую, очень специализированных задач. Это могут быть задачи обеспечения высокой доступности, мобильности, повышения надежности, наполнения хранилища, реакции на отказ и т.п. В отличие от этих задач, или в дополнение к ним, внедрение распределенной системы с репликацией вызвано такими причинами, как необходимость снижения трафика между узлами через внешние коммуникационные соединения, для балансирования нагрузки между узлами, для консолидации данных, для резервирования данных, для масштабирования и повышения доступности приложений баз данных, а также для локального кэширования данных пользователем.

Основной сложностью при организации репликации является распространение обновлений. Если один из участвующих в репликации объектов был изменен, это изменение нужно корректно выполнить на всех участвующих в репликации узлах.

Фундаментальным принципом распределенных баз данных, по определению Дейта, является то, что для конечного пользователя распределенная система должна выглядеть так же, как нераспределенная. Дейт, в своей книге «Введение в системы баз данных», сформулировал двенадцать основных целей, которые должна решать распределенная система, чтобы считаться идеальной с точки зрения репликации:

1. Локальная независимость.
2. Отсутствие опоры на центральный узел.
3. Непрерывное функционирование.

4. Независимость от расположения.
5. Независимость от секционирования.
6. Независимость от репликации.
7. Обработка распределенных запросов.
8. Управление распределенными транзакциями.
9. Аппаратная независимость.
10. Независимость от операционной системы.
11. Независимость от сети.
12. Независимость от типа СУБД.

Не все эти цели являются непременно обязательными для системы, претендующей на название системы с репликацией, но их достижение может принести такие преимущества, которые будут чрезвычайно полезны для распределенной системы. Давайте кратко рассмотрим каждую из этих целей и те выгоды, которые они дают. Более подробное их определение и объяснение можно найти в указанной выше книге Криса Дейта.

Локальная независимость означает, что все узлы распределенной системы должны быть независимы друг от друга и легко переводиться в автономный режим работы. Кроме того, все операции в рамках узла должны быть подконтрольны этому узлу и не зависеть от состояния или иным образом от других узлов распределенной системы. При этом локальные данные должны храниться и обслуживаться на своем узле в локальной базе данных, подчиняться действующим на этом узле правилам и ограничениям, быть доступными в рамках политики безопасности, действующей на локальном узле и в локальной базе данных, иметь локальное управление, аудит и администрирование. При обращении к данным из других узлов распределенной системы, данные должны сохранять свою локальную принадлежность.

На практике, во многих современных СУБД, каждая используемая в распределенных системах база данных, управляется отдельно и независимо от других баз данных, как будто бы каждая такая база никак не связана через сеть с другими базами данных. Хотя каждая база данных может работать с другими, они представляют собой отличные, отдельные системы, каждая из которых требует индивидуального подхода.

Поскольку данные реплицируются в место локальной обработки, то сбои в работе сети и удаленных систем оказывают незначительное влияние на пользователя. Данные могут обрабатываться локально, пока связь с источником данных не будет восстановлена.

Каждый узел системы с репликацией практически независим от других при выборе конфигурации и способа обработки данных. Можно произвольно выбирать тип и подмножество данных, которое будет передаваться или приниматься каждым из узлов. Допускается даже изменять оригинальные имена таблиц и столбцов издателя на свои, действующие

локально на подписчике. Кроме того, оптимизация структуры и доступа к данным каждого узла может выполняться индивидуально, исходя из его потребностей и конфигурации.

Дейт считает, что отсутствие опоры на центральный узел, в общем случае означает, что локальный узел не должен зависеть от служб, работающих на другом, возможно едином для всех узлов распределенной системы, «центральном» или «главном» узле. Здесь не имеются в виду службы, отвечающие за тиражирование данных, не должно быть служб централизованной обработки запросов или управления транзакциями, как и любых других служб, из-за которых система в целом может стать зависимой от центрального узла. В идеальном случае, каждый узел может быть самодостаточен, а распределенная система реализует схему одноранговой репликации с совмещением на каждом узле ролей издателя и подписчика. Однако, это не исключает возможности построения схем с централизованным тиражирование или накоплением данных. Часто требуется как раз централизованная схема управления распределенной системой, а опора на центральный узел задается как одно из бизнес-требований. К таким системам можно отнести хранилища баз данных, которые пополняются информацией из множества источников распределенной сети узлов, или наоборот, когда одна и та же информация тиражируется по узлам филиальной сети. Часто, единую точку входа создают в целях балансирования нагрузки на систему распределенных узлов, что тоже не укладывается в обозначенную Дейтом цель – отсутствие опоры на центральный узел.

Отсутствие опоры на центральный узел важно в тех случаях, когда все узлы должны быть равноправны и потеря любого не должна приводить к потере работоспособности остальных. Обычно, репликация выполняется отдельными от ядра СУБД модулями, специально предназначенным для репликации данных.

В своей книге Дейт под непрерывным функционированием понимает такие показатели СУБД как надежность и доступность. Часто, репликация используется не только для приближения данных к потребителю (повышение доступности), но и для повышения надежности хранения информации (за счет ее тиражирования). Под надежностью, в частности, понимается высокая степень вероятности того, что система будет работать в любой момент времени. Для распределенных систем с репликацией характерно сохранение работоспособности даже в случаях отказов части их компонентов, таких как отдельный узел. Под доступностью понимается вероятность непрерывного функционирования системы в течение заданного времени. Поскольку репликация позволяет дублировать данные, можно незаметно переключать пользователей между узлами, повышая тем самым доступность распределенных систем за счет дублирования.

Важно также отметить, что непрерывность функционирования важна с точки зрения одного узла. В большой распределенной системе может

постоянно существовать несколько отключенных или неработающих по каким-либо причинам узлов, и это может быть обычным явлением. Поэтому, одной из важнейших характеристик распределенной системы является возможность восстановления работоспособности отключенного узла или группы узлов, причем, вне зависимости от того, какие функции в распределенной системе эти узлы выполняют. Должны предусматриваться такие процедуры, как возобновление тиражирования с обменом накопленных изменений или с повторной, полной инициализации данных узла.

На непрерывность функционирования локального узла распределенной системы могут влиять не только факторы репликации, но и другие присущие СУБД факторы, такие как необходимость периодического обслуживании, профилактика и т.п. Возможности распределенной системы можно использовать и для того, чтобы сохранить пользовательский доступ к данным даже во время технологических окон обслуживания узла, переключив их на другой, равнозначный узел.

Основная идея независимости от расположения состоит в том, что с точки зрения пользователя любые доступные ему в распределенной системе данные должны выглядеть как хранящиеся на локальном узле. Пользователь ничего не должен знать о месте физического хранения данных, а приложения смогут обращаться к таким данным унифицировано и одинаково работать на любом из узлов распределенной системы, даже если данные будут перемещены на другой узел.

Независимость от расположения позволяет размещать данные там, где они нужны. Это уменьшает зависимость от отказов сетей передачи данных и сокращает общие расходы на обмен информацией.

Система, по словам Дейта, независима от секционирования, если секционирование (т.е. физическое разделение таблиц на части или фрагменты) выполняется прозрачно для конечного пользователя, т.е. логика работы должна быть такой же, как и с несекционированными данными.

Секционирование используется для повышения производительности извлечения данных за счет сокращения объема оперируемых данных. Также, с помощью распределенного секционирования данные могут физически храниться в том месте, где они чаще всего используются. Это позволяет локализовать большие запросы и сократить объем внешнего сетевого трафика. Применение распределенного секционирования позволяет добиться совместного использования удаленными узлами только тех данных распределенной системы, в которых они нуждаются, и не более.

Независимость от секционирования – это гарантия того, что секции данных в любой момент могут быть заново собраны в одну таблицу, причем это не повлияет на работоспособность и правильность отображения данных в приложениях пользователя.

Независимость от репликации означает, что репликация должна быть «прозрачной для пользователя», т.е. логика работы с данными должна быть такой же, как и без репликации. Иными словами, таблицы или секции таблиц базы данных, с которыми работает пользователь, могут вклю-

чаться в репликацию без переделки приложений и, зачастую, незаметно для пользователя. Эта цель не так проста, как кажется на первый взгляд, поскольку «прозрачность» репликации зависит не только от реализации репликации производителем СУБД, а также от того, как была реализована топология и схема репликации объектов базы данных разработчиком приложения или администратором баз данных.

Кроме того, отвечающие за репликацию компоненты могут влиять на работу узла и косвенно влиять на операции пользователей, что тоже усложняет решение поставленной цели. Непросто обеспечить независимость от репликации в условиях гетерогенных узлов.

Распределенные системы должны быть реляционными, т.к. такие системы позволяют оптимизировать обработку запросов. По производительности распределенная реляционная система может существенно превосходить не реляционную систему.

Для распределенных запросов характерно использование протокола двухфазной фиксации, включая гетерогенные источники, а также возможен асинхронный режим работы посредством очередей. Подключение гетерогенных источников через своего провайдера позволяет использовать оптимизатор гетерогенного источника.

Дейт подчеркивает два основных аспекта управления транзакциями: управление восстановлением и управление параллельностью работы. В репликации оба этих аспекта получают дополнительный смысл и значение.

Когда мы имеем дело с распределенной системой, в рамках одной транзакции могут быть выполнены DML или DDL-операции с объектами на нескольких узлах, причем, распределенная система должна уметь управлять ходом исполнения всех таких операций и не допускать блокирование одних и тех же ресурсов при исполнении операций в рамках распределенной транзакции. Точно также, репликация должна гарантировать атомарность распределенных транзакций и предоставлять возможность отката таких транзакций, как это предусмотрено протоколом двухфазной фиксации.

Управление параллельностью у распределенных систем на основе SQL Server основано на механизмах блокировок. В случае гетерогенных узлов, система обеспечивающих распараллеливание запросов блокировок должна быть транспарентной.

В современных СУБД передача транзакций может осуществляться асинхронно, когда узел-получатель находится в офлайне или в режиме отложенных команд, и когда агенты репликации связываются асинхронно, помещая запросы в очереди и затем продолжая их работу. В некоторых СУБД наряду с двухфазной фиксацией, используется механизм репликации событий, который обеспечивает оповещение удаленных узлов о событиях, позволяя каждому такому узлу реагировать на события в соответствии с его потребностями.

Под аппаратной независимостью подразумевается возможность запуска одной и той же СУБД на разных аппаратных платформах. Жела-

тельно, чтобы работающие на разных платформах узлы в рамках распределенной системы с репликацией были равноправны.

Уже давно востребована возможность запуска одной и той же СУБД под управлением разных операционных систем. Сегодняшние возможности виртуализации во многом снижают сложность решения этой задачи.

Подразумевается, что распределенная система с репликацией должна уметь поддерживать используемые для связи между узлами коммуникационные сети и не зависеть от замены сети или коммуникационного канала, а также маршрута передачи данных.

Под независимостью от типа СУБД подразумевается, что системы управления базами данных, которые могут работать на разных платформах и под управлением разных операционных систем, при включении в распределенную систему смогут использовать для организации репликации данных один и тот же интерфейс.

Выбор типа репликации

Одними из основных факторов при выборе типа репликации для тиражирования данных является степень автономности подписчиков и величина допустимой задержки передачи изменений от породившего их узла на все участвующие в репликации узлы. В SQL Server существует три разновидности репликации: репликация моментальных снимков, репликация транзакций и репликация слиянием. У всех этих типов есть подтипы, со своими характеристиками автономности и задержки. Под автономностью принято понимать возможность использования базы данных без обеспечивающих репликацию механизмов, и без подключения к другим системам баз данных. Например, база данных на портативном компьютере может использоваться и тогда, когда компьютер не подключен к сети. Задержка же определяется периодом времени, в течение которого расположенные на разных узлах копии данных могут быть не идентичны из-за незавершенности процесса тиражирования изменений. Это могут быть секунды, минуты, часы, дни и даже месяцы, максимальное время возможной задержки определяется требованиями бизнес-правил. Кроме репликации, у SQL Server есть еще несколько технологий, пригодных для тиражирования данных, для которых тоже применимы характеристики автономности и задержки. К таким технологиям можно отнести: распределенные запросы, технику резервирования с последующим восстановлением на другом узле, доставку журналов, зеркальное отражение, копирование DTS-пакетов средствами службы Integration Services, экспорт/импорт через BCP, а также новую технологию в Visual Studio 2008 – службы синхронизации для ADO.NET. Многие из этих технологий применяются на разных стадиях репликации.

С точки зрения автономности, многие технологии близки по показателям и легко объединяются в группы. Так, например, все способы передачи данных посредством файлов импорта/экспорта показывают хорошую ав-

тономность, поскольку в перерывах между передачами и загрузкой/выгрузкой данные доступны для локального использования. Однако, при передаче больших объемов данных посредством транспортных файлов может сильно возрасти время, затрачиваемое на их транспортировку и загрузку/выгрузку. При этом задержка тиражирования может быть весьма значительной. В репликации аналогичные показатели автономности наблюдаются у двух типов репликации. Во-первых, это репликация слиянием, которая имеет в своей основе триггерный механизм, не использует распределенных транзакций и нуждается в метаданных других узлов только во время сеансов обмена изменениями с издателем. Все это делает участвующие в репликации узлы автономными и степень автономности высока. Этот тип репликации отличается небольшими значениями задержки тиражирования. Кроме того, типы репликации, допускающие передачу изменений с подписчика на издателя (а репликация слиянием является по своей природе двунаправленной с точки зрения передачи таких изменений) имеют специальные параметры запуска своих агентов репликации, позволяющие регулировать число изменений в рамках одного сеанса. Таким образом, можно дифференцированно влиять на задержку тиражирования изменений, передаваемых с издателя каждому подписчику. Все добавления и изменения данных записываются в специальные локальные таблицы метаданных репликации слиянием. Во время сеансов синхронизации агент репликации слиянием сверяет метаданные подписчика и издателя, выполняя предписываемые метаданными действия по синхронизации информации обоих источников. Поскольку одни и те же данные могут изменяться одновременно на разных узлах и целостность ключей должна обеспечиваться во всей распределенной топологии репликации, в репликации слиянием возможно возникновение конфликтов изменений данных, и для разрешения этих конфликтов предусмотрены специальные меры и механизмы.

Второй из наиболее автономных типов репликации – это репликация моментальных снимков. В чистом виде, репликации моментальных снимков использует для тиражирования данных их моментальные снимки, после чего распространитель тиражирует эти снимки (набор файлов или архивный файл) подписчикам, которые потом их самостоятельно применяют в своих базах данных. Создание и тиражирование моментальных снимков может осуществляться на периодической основе или по требованию. Поскольку в самом простом случае каждый подписчик получает одни и те же данные и не может вносить в них изменения, конфликты исключены. Получается, что каждый подписчик имеет свою копию публикации издателя, чем обеспечивается очень высокая автономность, а возможность тиражирования изменений данных на стороне подписчика не предусмотрена вовсе. Время передачи снимка от издателя подписчикам определяется размерами транспортных файлов, характеристиками канала передачи данных и возможностями серверного комплекса и может быть довольно большим, обуславливая высокую

задержку тиражирования. Т.к. в репликации моментальных снимков используется передача всех реплицируемых объектов и данных, этот тип репликации, наряду с восстановлением из резервной копии, используется во всех других типах репликации для первоначальной синхронизации данных. Существуют разновидности репликации моментальных снимков, которые предусматривают возможность обновления реплицируемых данных на подписчике. Например, в SQL Server 2000 такое обновление осуществлялось посредством распределенной транзакции непосредственно в базе данных издателя или через механизм обслуживания очередей. Нужно понимать, что использование распределенных транзакций или очередей влияет на автономность базы данных подписчика и, в случае интенсивного и постоянного использования этих механизмов, может привести к потере автономности подписчика.

Репликация транзакций дает не больше возможностей автономности для подписчиков, чем репликация моментальных снимков с обновлением на подписчике. Этот тип репликации характерен тем, что данные, вносимые, изменяемые или удаляемые на издателе, тиражируются подписчикам в виде хронологически выстроенных транзакций. Поскольку каждый подписчик получает одни и те же данные и в простом случае не вносит в эти данные изменений, возникновение конфликтов исключается. Между сеансами синхронизации каждый подписчик обладает целостным снимком данных (автономность в некотором роде). Поскольку транзакции доставляются подписчикам не мгновенно, изменения в данных доступны на подписчике тоже с некоторой задержкой тиражирования. Возможен вариант репликации транзакций с обновлением на подписчике, природа которого очень похожа на то, как это реализовывалось в репликации моментальных снимков с обновлением на подписчике. Этот вариант накладывает еще больше ограничений на автономность подписчиков.

Агенты репликации

Одну из ведущих ролей в репликации SQL Server играют агенты, которые являются обычными программами, реализованными в виде исполняемых модулей. Наиболее важную роль в репликации играют четыре агента репликации, это программы: distrib.exe – Replication Distribution Agent (агент распространителя); snapshot.exe – Replication Snapshot Agent (агент моментальных снимков); replmerg.exe – Replication Merge Agent (агент слияния); и logread.exe – Replication Log Reader Agent (агент чтения журнала транзакций в репликации транзакций).

Для SQL Server 2000 все эти четыре программы можно найти в каталоге по умолчанию: «C:\Program Files\Microsoft SQL Server\80\COM». Для SQL Server 2005/2008 путь к программам отличается только папкой версии, вместо «80» будет «90» или «100». Для того чтобы посмотреть параметры вызова этих программ, необходимо запустить соответствующие исполняемые файлы с ключом «-?». На экране будет представлен син-

таксис их запуска и перечень возможных ключей. Эта информация подробно изложена в документации, поставляемой с сервером баз данных.

Обратите внимание на то, что параметры могут быть определены в любом порядке. Когда дополнительные параметры не определены, используются значения предопределенных параметров в системном реестре локальной системы.

Большинство параметров определяют метаданные, необходимые для подключения программ, однако целый ряд параметров применим для тонкой настройки производительности репликации, а также для регулировки и настройки поведения алгоритмов, заложенных в работу агентов, на разных этапах логической последовательности операций. Окружение участвующих в репликации серверов и возможности их коммуникаций могут сильно отличаться на разных участках топологии репликации. Для учета возможных отличий и ограничений можно использовать существующие параметры вызова программ-агентов репликации. В табл. 1 представлены параметры вызова агентов репликации SQL Server 2000, с помощью которых можно регулировать порождаемую репликацией нагрузку на серверы, а также ограничивать трафик репликации между серверами.

Табл. 1. Управление нагрузкой и трафиком

BcpBatchSize	MaxBcpThreads	ReadBatchThreshold
Buffers	MaxCmdsInTran	SrcThreads
CommitBatchSize	MaxDeliveredTransactions	StartQueueTimeout
CommitBatchThreshold	MaxDownloadChanges	UploadReadChanges-PerBatch
DestThreads	MaxUploadChanges	UploadReadChanges-PerBatch
DownloadGenerations-PerBatch	PacketSize	UploadWriteChanges-PerBatch
DownloadReadChanges-PerBatch	PollingInterval	UseInprocLoader
DownloadWriteChanges-PerBatch	ReadBatchSize	

Набор параметров запуска программы агента репликации может быть уникальным для каждого сервера или единым для всех серверов-подписчиков. Задавать параметры можно через стандартный или пользовательский профиль агента, а также непосредственно в задании по расписанию, которое обслуживается агентом сервера и в котором одним из шагов задания вызывается программа агента репликации.

Параметры запуска агента распространителя можно изменить, добавив к стандартному набору параметров дополнительный, например, с именем UseInprocLoader и присвоив ему значение «1». Для справки замечу, что это повышает эффективность использования первоначально-

го снимка, предписывая агенту распространителя использовать команду BULK INSERT при применении файлов снимка на подписчике.

Профиль агентов репликации может быть задан, переопределен или создан с помощью специализированного мастера, в SQL Server 2000 он носил название «Agent Profiles for».

Мастер использует в своей работе системные таблицы в базе данных msdb. В таблице MSagent_profiles храниться информация о существующих профилях агентов репликации. Посмотреть эту информацию можно с помощью следующего запроса:

```
SELECT [profile_id]
    ,[profile_name]
    ,CASE [agent_type]
        WHEN 1 THEN 'агент моментальных снимков'
        WHEN 2 THEN 'агент чтения журнала'
        WHEN 3 THEN 'агент распространителя'
        WHEN 4 THEN 'агент слияния'
        WHEN 9 THEN 'агент чтения очереди'
    ELSE ''END
    ,[type]
    ,ISNULL ([description], '') AS description
    ,[def_profile]
FROM [msdb].[dbo].[MSagent_profiles]
ORDER BY [profile_id]
```

Для каждого из типов репликации существует заранее предопределенный набор стандартных системных профилей. Воспользовавшись выбором соответствующего режима, можно применить указанный профиль ко всем агентам репликации, зарегистрированным на сервере.

Каждый профиль имеет заданный набор параметров, которые в электронной документации к SQL Server принято называть аргументами. По умолчанию, набор параметров такой, как это определено в таблице MSagent_parameters. Однако, параметров может быть больше, чем определено и полный список возможных аргументов для каждого из типов агентов можно увидеть в таблице MSagentparameterlist.

С помощью параметров можно влиять на производительность работы агентов, а, следовательно, и на репликацию в целом. Однако, стоит отдавать себе отчет в том, что публикация только необходимого количества данных может оказаться самым эффективным методом повышения производительности репликации.

Работой агентов репликации управляет служба SQL Server Agent. Это не только позволяет обеспечить простоту запуска агентов репликации по расписанию, практически, в стандартной реализации все агенты репликации работают как подсистемы службы SQL Server Agent. Однако, вы можете начать сеанс репликации, запустив соответствующего агента репликации из командной строки, из прикладной программы, из командлета PowerShell или, например, из командного файла, вызываемого на

исполнение планировщиком операционной системы. В SQL Server 2000 были реализованы компоненты для контроля работы агентов репликации из пользовательских приложений посредством Microsoft ActiveX controls. Соответствие библиотек программам агентов репликации показано в табл. 2.

Табл. 2. Библиотеки Microsoft ActiveX controls

Имя агента	Исполняемый файл	Элемент управления ActiveX
Агент моментальных снимков	snapshot.exe	sqlinitx.dll
Агент чтения журнала	logread.exe	Нет
Агент чтения очередей	qrdrsvc.exe	Нет
Агент распространителя	distrib.exe	sqldistx.dll
Агент слияния	replmerg.exe	Sqlmergx.dll

В SQL Server 2005 был сделан следующий шаг по улучшению возможностей управления работой агентов репликации из приложений и встраивания репликации в приложения третьих фирм. Одним из новшеств репликации в этой версии стал набор объектов управления репликацией, который называется: «Replication Management Objects» (RMO), и представляет собой управляемый код, предоставляющий доступ к функциональности агентов репликации. Все задачи репликации, которые доступны в программе SQL Server Management Studio, могут выполняться программно с помощью RMO.

У репликации и ее агентов есть свои собственные счетчики производительности, которые можно найти в оснастке системного монитора. Например, число запущенных агентов репликации показывает счетчик: «Microsoft SQL Server: Replication Agents».

Давайте рассмотрим четверку наиболее распространенных в использовании агентов репликации, делая упор на параметры их запуска, настройка которых может помочь в регулировке порождаемой агентом нагрузки на сервер или в ограничении порождаемого работой агента трафика в сети.

Replication Snapshot Agent

Программа агента моментальных снимков (Replication Snapshot Agent) на основе заданной параметрами запуска или профилем конфигурации создает файлы моментального снимка. Моментальный снимок содержит схему данных и сами данные изданных таблиц и объектов базы данных. Файлы сохраняются в папке моментальных снимков, а запуск агента выполняется по расписанию соответствующим заданием на сервере, где расположена база данных распространителя. Кроме того, агент моментальных снимков фиксирует в базе данных распространителя информацию о состоянии синхронизации. Агент моментальных снимков не пред-

назначен для распространения созданных им снимков. При репликации снимками этот агент запускается настолько часто, насколько необходимо обеспечить приемлемую периодичность обеспечения синхронности подписанных баз данных. Для репликации транзакций или репликации слиянием моментальные снимки необходимы реже, вплоть до того, что они могут создаваться только тогда, когда возникнет необходимость подключения нового подписчика.

После того, как запущенный на распространителе агент моментальных снимков устанавливает подключение к издателю (по умолчанию, это может быть тот же самый сервер), он устанавливает совмещаемую (shared lock) блокировку на всех таблицах, включенных в публикацию. Это ему нужно для копирования схемы данных и гарантии непротиворечивости снимка. Поскольку такой уровень блокировки мешает другим пользователям вносить изменения в таблицы, создание снимка лучше планировать на время минимальной активности. Сценарии схемы таблиц, представлений и хранимых процедур сохраняются в файлах с расширением «sch». Сценарии создания индексов сохраняются в файлах с расширением «idx». Сценарии создания триггеров будут в файлах с расширением «trg». Ограничения для поддержки декларативной ссылочной целостности будут находиться в файлах с расширением «dri». Данные будут находиться в файлах с расширением «bcp». Агент моментальных снимков получает всю необходимую информацию в базах данных издателя, выгружая данные из публикуемых таблице в файлы моментального снимка. Эти файлы представляют собой сценарии для создания всех необходимых объектов и синхронизации данных, сохраняя моментальный снимок данных на момент времени создания снимка. Снимок создается для тех статей, которые входят в публикацию, и все это регистрируется в системных таблицах базы данных распространителя: MSrepl_commands и MSrepl_transactions. В таблице MSrepl_commands хранятся указатели на места размещения снимков статей, а также ссылки на предваряющие синхронизацию сценарии, если таковые имеются. В таблице MSrepl_transactions хранятся ссылки на соответствующие задачи синхронизации подписчика. После того как записи в этих двух таблицах будут сделаны, совмещаемая блокировка с издаваемых таблиц снимается.

В SQL Server 2000 мастер создания публикации «Create Publication Wizard» по умолчанию не включает опцию параллельной работы со снимком, которая позволяет смягчить воздействие, вызванное наложением совмещаемой блокировки на издаваемые таблицы. В свойствах публикации, на вкладке Snapshot можно включить опцию «Do not lock tables during snapshot generation», которая позволяет пользователям публикуемой базы данных продолжать работу и во время создания моментального снимка. Не рекомендуется включать эту опцию, если публикуемая таблица имеет уникальный индекс, который не является первичным ключом или кластерным индексом. Кроме того, включение этой опции может существенно увеличить время создания снимка.

Для создания снимков данных используется программа массового копирования BCP. В профиле агента создания снимков и в числе аргументов вызова программы агента моментальных снимков есть параметры, которые позволяют оптимизировать работу BCP. В частности, речь идет о значение аргумента BcpBatchSize, который позволяет разбивать поток экспорта/импорта данных на блоки. Этот аргумент задает число строк, используемых в операциях массового копирования. При выполнении операций «BCP IN», указываемый размер блока – это число строк, которые посылаются серверу как одна транзакция, а также число строк, которые должны быть посланы, чтобы агент распространителя зарегистрировал в своем журнале очередной шаг в последовательности операций BCP. При выполнении операций «BCP OUT» используется установленный по умолчанию размер пакета, равный 1000. Значение 0 соответствует отсутствию регистрации шагов BCP. Если вы сомневаетесь, какой размер блока следует выбрать для создания моментального снимка вашей публикации, воспользуйтесь значением, предлагаемом по умолчанию в системном профиле агента моментальных снимков. Для этого агента в SQL Server 2000 предлагался только один профиль, на основании которого можно строить свои, пользовательские профили для всех публикаций или для каждой публикации в отдельности.

Табл. 2. Предопределенный в SQL Server 2000 профиль для агента моментальных снимков

Параметр	Значение по умолчанию
BcpBatchSize	100000
HistoryVerboseLevel	2
LoginTimeOut	15
MaxBcpThreads	1
QueryTimeOut	300

Заметное влияние на время создания снимка оказывает аргумент MaxBcpThreads. Он определяет число потоков операций массового копирования, которые могут быть выполнены параллельно. Максимальное число потоков и подключений, которые существуют одновременно, будет не больше, чем значение MaxBcpThreads или число запросов на массовое копирование, которые окажутся при синхронизации транзакций в базе данных распространителя (для агента слиянием они будут в системной таблице sysmergeschemachange базы данных издателя). Аргумент MaxBcpThreads должен быть больше нуля и не имеет верхнего предела. Значение по умолчанию равно -1. При применении снимка, который был создан издателем, использующим опцию параллельного создания снимка, будет использоваться один поток, независимо от заданного для аргумента MaxBcpThreads значения. Практика показывает, что даже для однопроцессорных систем есть смысл устанавливать значение аргумен-

та MaxBcpThreads больше единицы. На одном процессоре я предпочитаю задавать этому аргументу значение, равное 3.

Создание снимка может приводить к повышению нагрузки на дисковую подсистему, где располагаются файлы моментального снимка публикации. При планировании размещения файлов и разбиении дисковых массивов необходимо учитывать, что вызванная созданием снимков дополнительная нагрузка может негативно сказаться на других процессах и задачах, обращающихся к этим же дискам. Хорошей практикой является планирование создания моментальных снимков в часы наименьшей нагрузки, чтобы при необходимости использовать для передачи агенту распространителя уже заготовленный снимок. Такой подход возможен, если после применения снимка репликация не переданных с ним изменений будет выполняться с приемлемой нагрузкой аппаратных и коммуникационных средств, и не будет превышать допустимые регламентами сроки. Иначе, необходимо резервировать запас производительности процессоров и дисковой подсистемы для обеспечения необходимого уровня производительности во время создания моментальных снимков.

При каждом запуске агента моментальных снимков выполняется проверка на появление новых подписок. Если за время, прошедшее с последнего создания снимка, не появилось новых подписок, файлы сценариев или снимков данных не создаются. Однако, если была задана команда немедленного создания моментального снимка, агент моментальных снимков создаст новые файлы схемы и снимков данных.

Для облегчения работы с уже готовыми моментальными снимками рекомендуется использовать опцию сжатия снимка в CAB-файл. Для гарантии доставки файла снимка лучше снимки выкладывать в каталог сервера FTP. Однако, сжатие файлов снимков требует дополнительного расхода ресурсов системы при создании и применении файла моментального снимка и увеличению времени генерации и применения снимка.

Запуск программы агента моментальных снимков лучше производить только тогда, когда это необходимо, и не во время пиковой нагрузки, чтобы избежать блокировок. Папки моментальных снимков лучше размещать на диске, не используемом для хранения файлов базы данных или файлов журналов транзакций. Также, стоит использовать по одной папке для моментальных снимков каждой публикации, чтобы сократить количество операций для поддержки снимков в других папках.

В случае длительного прерывания сеансов репликации, может наступить момент, когда выгодней не дожидаться синхронизации всех накопленных изменений, а применить свежий моментальный снимок. При большом размере публикации это может оказаться весьма продолжительной и дорогостоящей операцией. Чтобы избежать такой ситуации, желательно настроить рассылку оповещений обо всех критичных событиях в системе репликации. Желательно также создать «оператора последней надежды» (fail-safe operator), чтобы гарантировать доставку таких уведомлений.

Применением моментального снимка на подписчике занимается агент распространителя. Для этого он вначале устанавливает подключение к серверу, где располагается база данных распространителя, чтобы получить в таблицах MSrepl_commands и MSrepl_transactions информацию о том, какие снимки необходимо применить данному подписчику. Там же он получает сведения о расположении папки моментального снимка. Следующим шагом он применяет в базе данных подписчика сценарии создания/изменения схемы, которые создают все необходимые объекты. При необходимости осуществляется конвертация типов данных, которая бывает нужна для подписчиков других версий или СУБД. Когда схема готова, осуществляется операция массовой вставки данных. После того как все статьи станут синхронны с публикацией, и для основных таблиц будет обеспечена транзакционная и ссылочная целостность, в подписанной базе данных создаются системные объекты репликации, а также триггеры, процедуры или представления. Все эти шаги регистрируются в базе данных распространителя.

Для мониторинга производительности работы агентов моментальных снимков можно использовать счетчики производительности объекта «SQL Server: Replication Snapshot.», которые показывают сколько команд или транзакций было передано распространителю. Счетчик: «Snapshot: Delivered Cmds/sec» показывает передаваемое распространителю число команд в секунду, а счетчик «Snapshot: Delivered Trans/sec» передаваемое в секунду число транзакций.

Replication Distribution Agent

Агент распространителя (Replication Distribution Agent) используется для доставки моментальных снимков подписчикам, а также для тиражирования изменений в репликации транзакций. Свою конфигурацию он получает из строки запуска или запрашивает ее из заданного ему профиля. После этого он перемещает моментальный снимок (для репликации снимков и репликации транзакций), определенный в таблицах базы данных распространителя (для репликации транзакций), в указанные места назначения на подписчиках. Агент распространителя запускается для каждой публикации и исполняется на подписчике, при подписке в режиме «pull», а при подписке в режиме «push», он работает на распространителе. Агент распространителя умеет передавать данные в виде команд или транзакций не только напрямую подписчикам, но и на вход DTS-пакета. Для репликации моментальных снимков периодичность запуска определяется требованиями к обновлению информации на подписчиках. Для репликации транзакций передача моментального снимка выполняется при первоначальной синхронизации (если в настройках подписки не указано иное) или для принудительной, повторной инициализации подписчика. В остальное время агент распространителя в репликации транзакций занимается доставкой подписчикам пакетов команд изме-

нения данных, которые он берет из своей базы данных, обращаясь к таблице MSrepl_commands, содержащей записи о реплицированных командах, и к таблице MSrepl_transactions, которая содержит по одной строке для каждой реплицируемой транзакции. Кроме того, распространитель передает подписчикам хронологию сеансов репликации и сведения о возникших ошибках.

SQL Server 2005 появились два новых типа первоначальной синхронизации (один задавался параметром «replication support only», а второй использовал инициализацию резервной копией), упрощающие инициализацию подписчиков, которые позволяют выполнить синхронизацию вручную (в англоязычной документации такая синхронизация называется: «no-sync subscriptions»). Необходимость ручной синхронизации была вызвана задачами автоматизации подготовки базы данных подписчика для репликации, а также, с помощью ручной синхронизации можно существенно сократить время простоя, если использовать уже синхронную резервную копию.

В SQL Server 2008 появился еще один новый тип синхронизации – инициализация по LSN. Этот тип синхронизации используется для внутренних нужд в одноранговой репликации транзакций. Он походит на инициализацию резервной копией, отличаясь тем, что инициализация будет продолжена начиная с указанного LSN, т.е. от Распространителя в базу данных подписчика будут догружены те транзакции, которые были зарегистрированы после указанного номера виртуального журнала. Такой тип синхронизации очень удобен для сокращения времени восстановления подписчика после отказа, поскольку догрузка транзакций занимает, как правило, меньше времени, чем полная инициализация, а Агент Распространителя продолжит работу с того места, где он остановился из-за отказа Подписчика, или отказа Зеркального отображения.

База данных распространителя пополняется сведениями о вновь созданных моментальных снимках, которые поставляет агент моментальных снимков. Агент чтения журнала запускает на издателе системную процедуру sp_replcmds, возвращающую команды, из которых состоят помеченные для репликации транзакции. Эти команды хранятся в базе данных распространителя, который доставляет их всем подписчикам и отслеживает, были ли команды доставлены успешно за отведенное для этого время. Кроме этого, агент чтения журнала, с помощью системной хранимой процедуры sp_repldone, обновляет запись в журнале транзакций издателя, которая идентифицирует последнюю распределенную транзакцию сервера. Т.е. те транзакции, которые были успешно реплицированы всем зарегистрированным подписчикам, могут быть убраны из журнала в резервную копию, и занимаемое ими место в журнале регистрации транзакций публикуемой базы данных на издателе может быть высвобождено для других виртуальных журналов. Информацию для этого он также берет из базы данных распространителя, где находит данные об успешно реплицированных командах.

С помощью аргумента MaxDeliveredTransactions программы агента распространителя можно задать максимальное число «push» или «pull»-транзакций, примененных на подписчике в рамках одного сеанса синхронизации. Значение 0 указывает, что будет применено максимально возможное число транзакций. Другие значения могут использоваться на подписчике для того, чтобы сократить продолжительность синхронизации с издателем. Это позволяет управлять трафиком транзакций по сети, а также нагрузкой, порождаемой агентом распространителя на сервер запуска и на сервер, который обслуживает базу данных подписчика.

Нагрузкой на подписчике позволяют управлять еще два аргумента агента распространителя. Аргумент CommitBatchSize задает число транзакций, которые будут исполнены подписчиком прежде, чем будет исполнена инструкция COMMIT обрамляющей транзакции. Значение по умолчанию равно: 100. Аргумент CommitBatchThreshold задает число команд репликации, которые будут исполнены подписчиком прежде, чем будет исполнена инструкция COMMIT. Значение по умолчанию равно: 1000.

В базе данных распространителя присутствуют и другие таблицы, используемые для некоторых режимов репликации. Таблица MSrepl_backup_lsns нужна для синхронизации репликации транзакций с резервной копией. Таблица MSrepl_errors содержит информацию об ошибках агентов распространителя и слияния. Таблица MSrepl_identity_range предназначена для управления диапазонами идентификаторов. Таблица MSrepl_originators нужна обновляемым подписчикам. Таблица MSreplication_queuedtraninfo нужна для организации очередей команд. В таблице MSrepl_version хранятся данные о текущих версиях репликации. Таблица MSreplication_monitordata используется монитором репликации, и т.д.

Если вы запускаете службу SQL Server Agent под учетной записью локальной системы (значение по умолчанию), а не под учетной записью пользователя домена, служба сможет обращаться только к локальному компьютеру. Если агент распространителя, работа которого управляется службой SQL Server Agent, будет при этом использовать для доступа к экземпляру SQL Server режим аутентификации Windows, агент распространителя не сможет нормально работать. В SQL Server 2000, параметр по умолчанию для режима аутентификации – собственная аутентификация SQL Server.

У агента распространителя существует несколько предопределенных профилей. Для SQL Server 2000 их было четыре. Один профиль использовался по умолчанию и включал значения аргументов запуска агента распространителя для наиболее распространенных вариантов репликации. Этот профиль можно взять в качестве отправной точки для начала настройки агентов распространителя.

Существует также профиль для непрерывного режима работы агента распространителя, который, работая в тандеме с агентом чтения журнала, может в непрерывном режиме отслеживать и тиражировать изме-

нения от издателя подписчикам. Нужно отметить, что непрерывный режим тоже является дискретным, и эту дискретность в секундах определяет еще один аргумент агента распределителя PollingInterval.

Другим профилем является профиль для расширенной диагностики и хронологии сеансов репликации агента распространителя. Он определяет детализацию хронологии, регистрируемой во время исполнения операций агента дистрибутора, агента слияния или в течение исполнения операций с моментальным снимком. Основное отличие этого профиля от других – это значение аргумента HistoryVerboseLevel, который определяет объем информации, протоколируемой в журнале работы агента. Увеличение уровня детализации помогает быстрее локализовать проблему и принять соответствующие меры за счет получения более подробной хронологии сеанса репликации. Нужно заметить, что этот аргумент используется во всех агентах репликации. Возможны три значения для этого аргумента. В профилях по умолчанию для агента распространителя, агента моментальных снимков и агента чтения журнала для этого аргумента задано значение 1. При таком уровне детализации всегда обновляются предыдущие хронологические записи с таким же как у текущей задачи состоянием (startup, progress, success и так далее). Если не существует ни одной предыдущей записи с тем же самым состоянием, в отчет вставляется новая запись. Для агента слияния в профиле по умолчанию аргументу задано значение «2». При таком уровне детализации вставляются хронологические записи, если эти записи не являются сообщениями об отсутствии активности или сообщениями о выполняющихся долго заданиях, для которых происходит обновление предыдущих записей. Если аргументу задать значение «3», это приведет к тому, что новые записи в хронологический журнал будут вставляться во всех случаях, кроме тех, когда они являются сообщениями об отсутствии активности. Кроме этого аргумента для получения максимально возможного уровня детализации хронологии работы агентов репликации, можно использовать два аргумента, которые позволяют выводить хронологию во внешний файл. Включается такой режим журналирования добавлением аргумента Output, значение которого задает путь к файлу отчета работы агента и его имя. Если путь не указан, вывод осуществляется на консоль. Если указанное имя файла существует, записи добавляются в конец файла. Еще один аргумент, OutputVerboseLevel, определяет уровень подробности отчета, сохраняемого в файл. Если уровень подробности - 0, записываются только сообщения об ошибках. Если уровень подробности - 1, будут записаны все сообщения о результатах работы. Если уровень подробности - 2 (значение по умолчанию), будут записаны все сообщения об ошибках и о результатах работы, которые очень полезны для отладки.

Устанавливать уровень подробности хронологии работы агентов репликации больше нулевого стоит в тех случаях, когда проводится тестирование репликации, на время периодического контроля работы агентов или в целях отладки и разрешения проблем. Отказ от ведения хроноло-

гии сеансов репликации может дать прирост производительности сеанса репликации до 15%.

Последний профиль достаточно редко применяется на практике и предназначен для тех случаев, когда репликация использует службу Windows Synchronization Manager.

Следует помнить, что когда агент распространителя применяет моментальный снимок на подписчике, он блокирует входящие в подписку таблицы базы данных подписчика. Кроме того, агент распространителя блокирует эти таблицы на время применения блока команд, обрамленных транзакцией.

Во время применения агентом распространителя моментального снимка на подписчике можно задействовать средства повышения производительности массовых операций, которые применяются в программе BCP. Использование при запуске агента распространителя аргумента UseInprocLoader повышает эффективность использования моментального снимка, предписывая агенту распространителя при применении файлов снимка на подписчике использовать команду BULK INSERT. В результате, производительность применения снимка может получить прирост до 30 % (если нет проблем с установками порядка сортировки (collation)). Также до 30% повышение в производительности может быть достигнуто при применении аргумента MaxBcpThreads. Этот аргумент определяет число потоков операций массового копирования, которые могут быть выполнены параллельно. Его смысл точно такой, как и у одноименного аргумента агента моментальных снимков. Совместное применение аргументов UseInprocLoader и MaxBcpThreads может дать до 50% выигрыша в производительности.

Еще одним полезным аргументом для повышения производительности агента распространителя является аргумет Buffers. Он задает число буферов, доступных для асинхронных транзакций. Значение по умолчанию равно 2. Увеличение этого числа может способствовать повышению эффективности за счет сокращения листания (memory paging) памяти. Однако, большое число установленных буферов увеличивает объем памяти, выделяемой для обслуживания листания. Оценить эффективность изменений значений этого параметра можно путем оценки времени подключения подписчиков к дистрибутору при изменении значений параметра. Особенностью этого аргумента является то, что не во всех версиях SQL Server его можно добавить в системные таблицы профилей репликации. В SQL Server 2000 его можно было добавить только в виде аргумента вызова программы агента, в специализированном шаге задания по расписанию агента сервера.

Если вы обновляете какой-либо столбец таблицы базы данных, обслуживаемой Microsoft SQL Server 2000 Standard и Enterprise Edition, который является частью уникального или составного индекса, SQL Server осуществляет обновление как отложенное изменение. Отложенное изменение означает, что команда UPDATE будет передана подписчику как

пара операций: DELETE и INSERT. Отложенное изменение более подробно описано в статье базы знаний Microsoft: «Q238254 INF: UPDATE Statements May be Replicated as DELETE/INSERT Pairs». Возможны случаи, когда передача подписчикам изменений в виде пары команд DELETE и INSERT не отвечает требованиям бизнес-правил, которые, например, могут предписывать передачу подписчику действий триггеров при обновлениях. Именно для того, чтобы помочь правильно разрешить эту ситуацию, был введен флаг трассировки №8207, который появился в SQL Server 2000 Service Pack 1, и при использовании которого в репликации транзакций допускаются изменения одной командой. Обновление уникального столбца, которое затрагивает только одну строку (изменение, которое принято называть «singleton»), осуществляется как команда UPDATE, а не как пара операций DELETE и INSERT. Если изменения затрагивают несколько строк, изменение будет выполняться как пара команд DELETE и INSERT.

Вы можете задать ключ трассировки 8207 на сервере, исполняющем роль издателя, выполняя команду DBCC TRACEON (8207, -1) при каждом запуске SQL Server, или добавив к параметрам запуска сервера баз данных ключ -T8207. Ключ трассировки 8207 используется только в варианте репликации транзакций без обновляемых подписчиков.

При большом количестве подписчиков рассмотрите возможные меры по сокращению частоты копирования изменений на распространителя. В таких случаях может оказаться выгодным запускать агента распространителя реже (в пределах, допустимых бизнес-правилами), это может существенно сократить количество операций.

Использование подписок типа «pull» или анонимных подписок позволяет разгрузить издателя и/или распространителя. Аналогично действует использование опции удаленного запуска агента (Remote Agent Activation). Анонимные подписки не хранят информацию о подписке в базе данных распространителя.

И наконец, если сервер-издатель перегружен, но не может быть масштабирован до необходимого уровня производительности, можно переместить базу данных распространителя на другой, менее загруженный сервер.

Для мониторинга производительности работы агентов распространителя можно использовать соответствующие счетчики производительности объекта «SQL Server: Replication Dist.», которые показывают сколько команд и транзакций было передано подписчикам из базы данных распространителя указанным агентом распространителя. Таких счетчиков три. Первый счетчик: «Dist: Delivered Cmds/sec», он показывает число предаваемых подписчику команд в секунду. Второй счетчик: «Dist: Delivered Trans/sec», показывает число передаваемых в секунду транзакций подписчику. И третий счетчик: «Dist: Delivery Latency», показывает задержку исполнения транзакции на подписчике после ее отправки из базы данных распространителя в миллисекундах.

Replication Log Reader Agent

Агент чтения журнала транзакций (Replication Log Reader Agent) является наиболее важным и уязвимым элементом топологии репликации транзакций. У издаваемой базы данных не может быть больше одного агента чтения журнала. Чтобы понять его работу, стоит кратко описать схему взаимодействия компонент SQL Server в репликации транзакций.

В этом типе репликации отдельные транзакции, обрамляющие команды INSERT, UPDATE и/или DELETE, передаются от издателя подписчикам. После запуска программы агент чтения журналов считывает свою конфигурацию из системных таблиц базы данных распространителя, в частности, там он узнает, где располагается и как называется издаваемая база данных. Первым шагом при настройке подписки репликации транзакций выполняют первоначальную синхронизацию схемы и данных подписчика с издателем, для чего на подписчике применяется моментальный снимок публикации с издателя.

После синхронизации, единственный для публикации агент чтения журнала по установленному расписанию или непрерывно просматривает журнал транзакций издаваемой базы данных, выбирая в нем записи о транзакциях, отмеченных для репликации. Обо всех обнаруженных транзакциях, подлежащих репликации, агент чтения журнала делает записи в соответствующих системных таблицах базы данных распространителя, и заодно сохраняет в этой базе данных записи о хронологии своей работы. В таблице базы данных распространителя MSlogreader_history агент чтения журнала делает записи о последних прочитанных в сеансах репликации номерах виртуальных журналов (LSN). Это необходимо для того, чтобы в следующем сеансе репликации начать чтение с этого номера LSN. Само чтение осуществляется с помощью системной хранимой процедуры sp_replcmds, которая будет просматривать все записи в журнале с указанного ей LSN и до конца журнала, если не задан ограничивающий размер чтения аргумент ReadBatchSize. Агент чтения журнала отбирает только те транзакции, которые отмечены для репликации и если они удовлетворяют условиям фильтрации статьи (когда публикация использует фильтры). Найденные записи буферизуются и копируются пакетами в системную таблицу MSrepl_commands в базу данных распространителя. На этом этапе агент чтения журнала оценивает эффективность составляющих транзакцию команд и может внести изменения в их синтаксис, если это приведет к повышению эффективности. Например, команды могут быть преобразованы в реплицируемые вызовы процедур с заданными агентом параметрами, или в синтаксис может быть добавлено использование первичного ключа (присутствующего, но не заданного явно), а также в известных случаях команда UPDATE может быть преобразована в пару команд DELETE и INSERT. Но, несмотря на все возможные изменения, есть взаимно однозначное соответствие между транзакциями в журнале и записями команд в базе данных распространителя.

Агент распространителя передает подписчикам на исполнение все реплицируемые команды, которые были обнаружены агентом чтения журнала. Если команды в базе данных распространителя являются частью подписки, подлежащей преобразованию, агент распространителя передает команды в загрузчик данных DTS-пакета. Если это обычная подписка, агент распространителя применяет команды на подписчике уже как транзакции. Когда команда успешно исполнена на всех зарегистрированных подписчиках, она соответствующим образом помечается в базе данных распространителя. Агент чтения журнала, на основании получения информации об успешности репликации команд, снимает с записи в журнале отметку о том, что соответствующая транзакция подлежит репликации. Для этого используется системная хранимая процедура sp_repldone. Это важный момент, поскольку при полной модели восстановления, те виртуальные журналы транзакций, которые отмечены для репликации, не будут высвобождать место в журнале транзакций при выполнении команды BACKUP LOG, что может привести к существенному увеличению размера файла журнала транзакций издаваемой базы данных. Усечены могут быть только те записи журнала транзакций, которые не помечены для репликации. Хранение в базе данных распространителя команд, подлежащих репликации всем подписчикам (replicated transactions stay), осуществляется до истечения установленного периода хранения (retention period). Если существуют анонимные подписчики, команды будут храниться до истечения этого периода. Из-за этого установка большого периода хранения может потребовать увеличения размера базы данных распространителя и отводимого под папки моментальных снимков места на диске.

Если в базе данных издателя была зарегистрирована транзакция, состоящая из большого числа команд, на репликацию такой транзакции всем подписчикам может потребоваться много времени. Агент чтения журнала расщепляет каждую транзакцию на блоки команд, которые ее составляют. Характеристики этих блоков и количество реплицируемых блоков в течение одного сеанса или интервала доставки определяется значениями, установленными для аргументов запуска программы агента чтения журнала. Этим фактом можно воспользоваться, чтобы уменьшить нагрузку, порождаемую большой транзакцией на участников репликации.

Например, используя уже упоминавшийся выше аргумент ReadBatchSize можно задать максимальное число транзакций, читаемое из журнала базы данных издателя. Если же использовать аргумент ReadBatchThreshold, то можно задать не число транзакций, а число команд репликации, которые читаются из журнала транзакций, для репликации подписчикам. Если этот аргумент не определен, агент чтения журнала будет осуществлять чтение до конца журнала транзакций или с учетом значения аргумента ReadBatchSize (число транзакций).

Еще один аргумент MaxCmdsInTran появился после выхода SQL Server 2000 Service Pack 1. Он определяет максимальное число инструк-

ций, сгруппированных в транзакцию, которые агент чтения журнала запишет в базу данных распространителя. Как раз использование этого параметра позволяет агенту чтения журнала совместно с агентом распространителя делить большие транзакции (состоящие из большего числа команд) издателя на несколько транзакций меньшего размера. Применение этого параметра может уменьшить конкуренцию на распространителе и время ожидания репликации между издателем и подписчиком. Поскольку первоначальная транзакция применяется меньшими порциями, подписчик может обращаться к строкам большой логической транзакции издателя до того, как транзакция будет завершена, что в принципе нарушает атомарность транзакции. Значение этого атрибута по умолчанию не позволяет делить транзакции издателя.

Как правило, сервер базы данных издателя и сервер базы данных распространителя располагаются в одной подсети или даже являются одним и тем же сервером. Для повышения производительности доставки данных агентом чтения журнала из журнала транзакций издаваемой базы данных в базу данных распространителя, который находится в этой же подсети и соединен с сервером издателя высокоскоростным соединением, можно использовать аргумент программы агента чтения журнала PacketSize, с помощью которого задается размер сетевого пакета в байтах. Значение по умолчанию - 4096 (байт). Увеличение размера пакета, если это допустимо используемым сетевым оборудованием, может снизить нагрузку на сеть и сократить число возможных коллизий.

По умолчанию агент чтения журнала работает в непрерывном режиме. Если его запускать по расписанию, то этим тоже можно снизить нагрузку на журнал транзакций, особенно, если за этот ресурс большая конкуренция системных процессов.

Судя по всему, большинство реализаций используют непрерывную работу агента чтения журнала. Главным образом этот выбор сделан потому, что такой вариант предлагается по умолчанию. Изменение этой опции позволяет осуществлять репликацию по расписанию. В SQL Server 2000 набор аргументов запуска агента чтения журнала в шаге задания агента сервера выглядел так:

```
-Publisher [ИМЯ ИЗДАТЕЛЯ] -PublisherDB [ИМЯ ИЗДАВАЕМОЙ БД] -Distributor [ИМЯ
РАСПРОСТРАНИТЕЛЯ] -DistributorSecurityMode 1 -Continuous
```

Аргумент «Continuous» как раз и обеспечивает непрерывность работы агента чтения журнала. Убрав этот аргумент из числа аргументов вызова агента, вы можете запускать его по расписанию. Однако, если нагрузка на журнал невелика и, если бизнес-правила требуют частого запуска, лучше оставить непрерывный режим работы.

Еще одной причиной потери производительности агента чтения журнала может быть использование в статье горизонтальных фильтров, поскольку агент будет применять фильтр к каждой строке фильтруемой таблицы.

В SQL Server 2005 была добавлена новая схема репликации транзакций, которая на языке оригинала называется Peer-to-Peer (P2P), а в русской версии электронной документации SQL Server 2005 ее принято называть одноранговой репликацией транзакций. Основная ее идея - публикуемые статьи могут быть подписаны. Все узлы в одноранговой топологии равноправны, каждый является и издателем и подписчиком на одну и ту же публикацию, схемы на всех узлах идентичны. Есть механизм блокирования циклических изменений. Конфликты в версии SQL Server 2005 не отслеживаются, поэтому необходимо принимать меры, предотвращающие одновременную модификацию одних и тех же данных на разных узлах. Реализован механизм автоматической повторной синхронизации после восстановления одного из узлов из резервной копии. Изменения в восстановленной из резервной копии базы данных на время синхронизации с другими узлами запрещены. Резервная копия должна быть сделана после включения базы в одноранговую репликацию. К ограничениям новой схемы можно отнести невозможность использования фильтров строк и столбцов. Кроме того, не рекомендуется использовать свойство колонок «identity», т.к. придется вручную разделять диапазоны. Нельзя использовать издателей/подписчиков, не являющихся SQL Server . Нельзя использовать параметры публикации: «Queued» или «immediate updating». Нельзя использовать DTS-преобразования подписки. Нельзя использовать системную хранимую процедуру sp_addarticle и столбцы со свойством «timestamp». Нельзя использовать параметр агента чтения журнала: MaxCmdsInTran и параметр агента распространителя SubscriptionStreams.

Как показывает время, возможности агентов репликации увеличиваются с появлением новых версий и при добавлении или развитии функциональных возможностей.

Агент чтения журнала в SQL Server 2005 получил еще одну, очень интересную функциональную возможность. Теперь он может быть задействован в публикации серверов Oracle через распространителя на основе SQL Server. Репликация транзакций с издателем на сервере Oracle основывается на стандартной схеме репликации транзакций, но отслеживание изменений на Oracle и доставка их распространителю реализуется дополнительным промежуточным слоем. На издаваемые таблицы репликация создает триггеры уровня строки, срабатывающие на вставку, изменение и удаление (DML). Триггеры отслеживают операции DML и фиксируют их в специальных таблицах – журналах для каждой статьи, в порядке их исполнения. Вставка и удаление добавляют одну строку в журнал, а изменение – две строки (состояние до и после изменения). При изменении первичного ключа, триггеры срабатывают так, что фиксируются все связанные с ключом изменения. Агент чтения журнала считывает изменения в журналах и переносит их в базу данных распространителя, в таблицы MSrepl_commands и MSrepl_transactions. Подписчикам изменения тиражирует агент распространителя.

Для мониторинга производительности работы агента чтения журнала транзакций можно использовать счетчики производительности объекта «SQL Server Replication: Logreader», которые показывают сколько команд или транзакций было передано распространителю, а также сопутствующую этим процессам задержку. Счетчик: «Logreader: Delivered Cmds/sec» показывает число команд в секунду, а счетчик «Logreader: Delivered Trans/sec» число транзакций в секунду, которые передаются распространителю. Еще один счетчик: «Logreader: Delivery Latency» показывает текущее время, в миллисекундах, которое прошло с тех пор, как транзакция после применения на издателе была передана в базу данных распространителя.

Для разрешения проблем репликации транзакций и исследования ее функционирования по всей цепочке топологии репликации используются следующие системные хранимые процедуры и команды: sp_browsereplcmds, sp_replcounters, sp_replshowcmds, sp_repltrans, sp_replflush, sp_repldone, sp_replqueuemonitor, sp_replcmds и DBCC OPENTRAN. В упрощенном виде алгоритм можно описать следующим образом. После обнаружения проблемы в мониторе репликации, включите максимальный уровень детализации истории агентов чтения журнала и распространителя. Измените значение аргумента агента распространителя BatchCommitSize на 1, это позволит обрабатывать единовременно по одной транзакции или инструкции, которые могут состоять из нескольких команд. Потом, отключите работу заданий по очистке метаданных распространителя, чтобы избежать удаления информации, необходимой для поиска и устранения проблемы. После этого, перезапустите агентов чтения журнала и распространителя (если это необходимо). На подписчике убедитесь, что в таблице MSreplication_subscriptions присутствуют и верны записи исследуемой подписки. Запомните значение в колонке transaction_timestamp для агента распространителя этой подписки, которое будет нам указывать на последнюю транзакцию, поставленную распространителем подписчику. В таблицах базы данных распространителя присутствует колонка xact_seqno, которая соотносится с колонкой transaction_timestamp на подписчике (за исключением завершающих нулей). Далее, полученную информацию можно использовать для вызова системной хранимой процедуры sp_browsereplcmds @xact_seqno_start = “ <transaction_timestamp> “ которая вернет реплицируемые команды.

Большим подспорьем в разрешении проблем репликации транзакций может стать системная функция fn_dblog, которая позволяет заглянуть внутрь журнала регистрации транзакций. Ее можно использовать, опираясь на тот факт, что в журнале даже после прохождения контрольной точки остаются записи реплицируемых транзакций, которые еще не были помечены агентом чтения журнала, как успешно реплицированные. Т.о. можно определить, например, являются ли транзакции репликации причиной роста файла журнала транзакций. Посмотреть это не составляет труда, достаточно простого вызова:

```
SELECT * FROM fn_dblog(null,null)
```

Далее, остается только проанализировать возвращаемую представленной выше командой информацию. Помните, что команды одной транзакции имеют одинаковый идентификатор. Есть особые метки, которые могут помочь Вам ориентироваться в содержимом журнала. Например, следующие две метки отмечают начало и конец прохождения контрольной точки: LOP_BEGIN_CKPT и LOP_END_CKPT. Такие метки, как LOP_BEGIN_XACT, LOP_INSERT_ROWS и LOP_COMMIT_XACT отмечают транзакцию вставки данных в таблицу, а по колонке AllocationUnitName можно определить какой объект был задействован в транзакции.

Replication Merge Agent

Программа агента слияния (Replication Merge Agent) получает конфигурацию и умеет не только проводить сеансы репликации слиянием между базой данных издателя и базой данных подписчика, но и применяет моментальный снимок для первоначальной или повторной инициализации подписчика, если это не запрещено в настройках подписки. Этот агент универсален и является самым сложным из всех агентов репликации. Он обеспечивает репликацию слиянием инкрементных изменений данных, которые произошли на издателе и на подписчике после создания первоначального снимка или последнего сеанса синхронизации. При этом агент слияния урегулирует конфликты согласно установленных правил или с использованием заданного администратором алгоритма разрешения конфликтов.

Агенту слияния не нужен агент распространителя, и он почти не использует базу данных распространителя. Кроме того, у этого агента больше всего разнообразных профилей работы.

Поскольку репликация слиянием использует триггерный механизм отслеживания изменений, каждой строке сопоставляется глобальный уникальный идентификатор и номер поколения ее изменения. Фактически, транзакции базы данных издателя или подписчика преобразуются в поколения изменений данных, записи о которых сохраняются в системные таблицы метаданных слияния: MSmerge_contents, MSmerge_tombstone и MSmerge_genhistory. Такие таблицы создаются в участвующих в репликации базах данных во время оформления публикации или подписки. Кроме того, триггерный механизм является той причиной, почему одна таблица не может участвовать в двух публикациях репликации слиянием, поскольку логика работы триггеров была бы очень сложной, а конфликты изменений для нескольких публикаций было бы очень трудно отслеживать. Во время создания публикации репликации слиянием к таблицам может быть добавлен столбец с типом uniqueidentifier и с именем rowguid. Это происходит потому, что репликации слиянием необходим столбец, который уни-

кально идентифицирует строку в распределенной топологии репликации.

Такое возможное изменение схемы данных нужно учитывать, поскольку его можно использовать в собственных приложениях или внести необходимые правки в уже существующий код, чтобы добавление столбца не привело к ошибкам обращения к таблице.

Кроме столбца, для таблицы создается уникальный некластеризованный индекс по столбцу rowguid. И это еще не все изменения схемы. Для каждой таблицы, включаемой в публикацию, создается набор триггеров, имена которых начинаются со следующих префиксов: ins_, del_ и upd_, и заканчиваются глобальными уникальными идентификаторами. Именно эти триггеры выполняют синхронизацию, сохраняя в системных таблицах информацию о тиражируемых изменениях. Триггеры для команд INSERT и UPDATE сохраняют значения rowguid измененных и вставленных строк в таблице MSmerge_contents. Триггер для команд DELETE сохраняет метаданные об удаляемых строках в таблице MSmerge_tombstone. При этом, в отличие от репликации транзакций, каждое изменение строки перезаписывает предыдущие изменения в таблице MSmerge_contents, т.е. в этой таблице хранятся только самые последние изменение и именно последние изменения попадают в очередной сеанс репликации.

Агент слияния сравнивает содержимое таблиц MSmerge_contents подписчика и издателя. Ориентируясь по значениям столбца rowguid и по номеру поколения каждой из представленных там строк, он копирует новые строки и изменяет уже имеющиеся. Если одна и та же строка в сеансе числится измененной на издателе и на подписчике, регистрируется конфликт. Механизм поколений позволяет принимать решение о том, какие строки должны быть изменены без полного просмотра таблиц.

Двунаправленный характер передачи изменений и другие особенности репликации слиянием объясняют большее чем у других агентов число аргументов запуска программы агента слияния. Многие из аргументов несут такой же смысл и названия, как у других агентов, что объясняется универсальностью агента слияния.

Так же как и у других агентов, целый ряд аргументов может использоваться для балансировки нагрузки сеанса репликации, а также для выстраивания передачи очередности поколений, когда велика вероятность частого изменения одной и той же строки. Например, аргумент SrcThreads задает количество потоков на источнике, которые агент слияния использует для того, чтобы посчитать изменения на источнике. В течение сеанса репликации источником будет подписчик, когда выполняются операции с типом «upload», а издатель будет источником, когда выполняются операции с типом «download». Аргумент DestThreads задает число потоков, которые агент использует для передачи изменений с одного сервера на другой сервер, участвующий в репликации слияни-

ем. В зависимости от направления передачи изменений, в качестве принимающего сервера может выступать как издатель, так и подписчик. Аргумент UploadGenerationsPerBatch задает число поколений, которые будут обработаны в одном пакете при загрузке изменений от подписчика к издателю. Поколение представляет собой логическую группу изменений в статье. Значение по умолчанию для профиля, рассчитанного на надежные коммуникации между издателем и подписчиком, принимается равным 100. Значение по умолчанию для ненадежной связи - 1. Аргумент UploadReadChangesPerBatch задает число изменений, которые будут считаны в одном пакете при загрузке от подписчика к издателю. Аргумент UploadWriteChangesPerBatch задает число изменений, которые будут применены в рамках одного пакета при загрузке изменений от подписчика к издателю. Аргумент DownloadGenerationsPerBatch задает число поколений изменений, которые будут обработаны в одном пакете при их загрузке от издателя к подписчику. Аргумент DownloadReadChangesPerBatch задает число изменений, которые будут прочитаны в одном пакете при их загрузке от издателя к подписчику. Аргумент DownloadWriteChangesPerBatch задает число изменений, которые будут применены в одном пакете при их загрузке от издателя к подписчику.

Еще одним, очень интересным аргументом вызова программы агента слияния, является ForceConvergenceLevel, который задает агенту степень слияния. В электронной документации SQL Server 2000 Books Online он не был документирован. Синтаксис использования этого аргумента следующий:

```
-ForceConvergenceLevel 0 | 1 | ( 2 (Publisher | Subscriber | Both) )
```

Значение по умолчанию – 0, при этом используется стандартное слияние без принудительного сведения. Значение 1 задает принудительное сведение для всех поколений. Значение 2 задает принудительное сведение для всех поколений и корректирует неправильные поколения. Для этого режима важно, где неправильные поколения должны быть исправлены.

Аргумент ForceConvergenceLevel полезен в тех ситуациях, когда изменения не сходятся в течение сеанса слияния (проблема известна как несходимость репликации). Использование аргумента ForceConvergenceLevel помогает разрешению проблем в таких ситуациях, когда в исходной, точной копии, были пропущены некоторые поколения (generation). Также его можно использовать, когда значения столбца «lineage» в системной таблице MSmerge_contents стали логически неправильными. Когда аргументу присваивается значение 1, агент слияния (в течение закрытия актуальных, открытых поколений, и открытия новых поколений) делает изменения значений поколений, сохраняемых в системных таблицах MSmerge_contents и MSmerge_tombstone так,

чтобы любые пропущенные поколения были учтены. Это гарантирует, что любые изменения, представленные этими поколениями, имеют право на согласование в течение следующего сеанса слияния. Однако, существует одно ограничение, которое определяет, что поколение уже не может существовать в точной копии адресата. То есть в системной таблице MSmerge_genhistory уже не может быть строки для этого поколения в адресате.

Когда аргументу ForceConvergenceLevel устанавливается значение 2, агент слияния исполняет по описанному предварительно сценарию все поколения и также выполняет хранимую процедуру на точной копии, указанной для исправления («fix up») любых неправильных значений столбца «lineage» в системной таблице MSmerge_contents. Столбец «lineage» в системной таблице MSmerge_contents, это столбец с типом varbinary, который хранит информацию об уровне строки (versioning). Столбец «lineage» используется при отслеживании изменений в репликации слиянием и для обнаружения конфликтов, связанных со столбцом «colv1» в системной таблице MSmerge_contents. Неправильные значения в столбце «lineage» или «colv1» могут привести к неправильным изменениям.

Однако, важно понимать, что этот аргумент гарантированно не исправит проблемы несходимости. Возможны случаи, когда изменения были пропущены, но из-за других проблем. Обратите внимание, что речь идет о тех проблемах, которые известны на сегодняшний день. В дополнение, можно отметить, что использование в промышленной среде аргумента ForceConvergenceLevel, может повлиять на производительность. Для получения дополнительной информации об известных проблемах, которые невозможно обойти с помощью аргумента ForceConvergenceLevel, изучите следующие статьи базы знаний Microsoft: «Q304703 FIX: Pull Subscribers Experience Non-Convergence After Running sp_mergecleanupmetadata Against a Published Database», «Q304551 FIX: Merge Publishing with Vertical Filters Results in Nonconvergence» и «Q304222 FIX: Merge Replication Non-Convergence Occurs with Local Subscribers when Published Table has 32 or More Columns».

В репликации слиянием очень важно поддерживать разумный баланс объема хранимых на подписчиках и на издателе метаданных, сосредоточенных в таблицах MSmerge_contents, MSmerge_tombstone и MSmerge_genhistory. В SQL Server 2000 Service Pack 1 появился новый аргумент агента слияния, который отвечает за то, будет ли производиться автоматическая очистка устаревших метаданных с учетом периода задержки издателя. Этот аргумент называется MetadataRetentionCleanup и ему может быть присвоено два значения 0 и 1. Важность отслеживания и своевременной очистки метаданных очень важна. На рис. 1 показана диаграмма трафика сети, на которой видно, как изменился трафик в 15 часов после того, как была выполнена очистка метаданных.

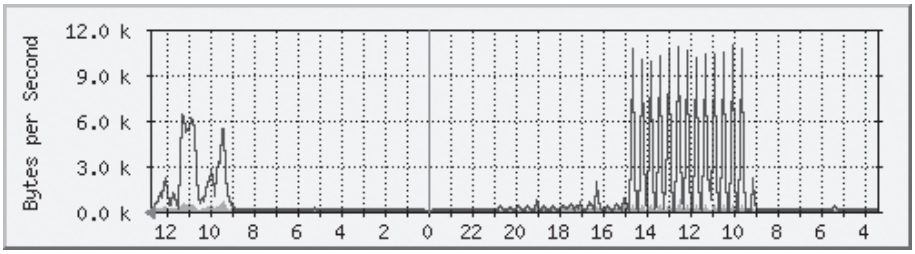


Рис. 1.

Это происходит потому, что во время сеанса слияния происходит сверка метаданных в базе данных издателя и подписчика. Именно по этим метаданным агенту слияния становится понятно, какие изменения нужно сделать в синхронизируемых базах данных. Фактически, для получения одного из наборов метаданных агенту слияния необходимо выполнить распределенный запрос и получить всю запрашиваемую выборку по трем таблицам с метаданными. Именно это объясняет такой высокий сетевой трафик на предыдущем рисунке. Снижение активного объема метаданных критично для подключений с низкими скоростями передачи. Кроме того, распределенные запросы к издателю могут долго удерживать необходимые блокировки, что будет осложнять жизнь другим агентам. По существу размер хранимых метаданных является ключевым фактором производительности репликации слиянием. Сложность учета воздействия объема метаданных на производительность заключается в том, что количество хранимых метаданных может увеличиваться постепенно и вырасти в проблему неожиданно для администратора баз данных.

В SQL Server 2005 появилась возможность организовать репликацию слиянием через веб-сервер, т.е. через протоколы http:// и https://. Это расширило возможности агента слияния и добавило ему новые, дополнительные аргументы запуска. Суть нового типа подключения репликации состоит в том, что изменения на подписчике упаковываются в формат XML и посылаются на веб-сервер (IIS) по протоколу HTTP. IIS посылает данные в бинарном формате издателю уже по принятому в интернет протоколу TCP/IP. На IIS сервере регистрируется библиотека replisapi.dll (SQL Server Replication Listener), которая обеспечивает обмен сообщениями с издателем и подписчиками. XML-данные обслуживаются библиотекой replrec.dll (Merge Reconciler), которая поставляется с SQL Server 2005. После сеанса слияния, Merge Reconciler вызывает библиотеку msgprox.dll (Message Replication Provider), которая в свою очередь вызывает xmlsub.dll (SQL Server Replication Provider) – обрабатывающую XML-файлы. Возможна подписка только по схеме «pull», поэтому агент слияния запускается на подписчике и может быть стандартным, «Merge Agent ActiveX control» или приложением с поддержкой Replication Management Objects (RMO). Добавлены новые параметры за-

пуска агента слияния: InternetURL, InternetLogin и InternetPassword. Моментальный снимок посылается по HTTP как вложенный файл. Синхронизация происходит непрерывно, по расписанию или по требованию. Возможно использование аутентификация IIS через «Secure Sockets Layers» (SSL), в дополнение к схеме «Basic Authentication». Также, в программе SQL Server Management Studio стало возможным подписать базы данных SQL Server Compact, создавая подписку на рабочем столе или непосредственно на мобильном устройстве.

Для мониторинга производительности работы агента слияния можно использовать счетчики производительности объекта «SQL Server: Replication Merge», которые показывают сколько строк было передано между издателем и подписчиком, и сколько при этом возникло конфликтов. Счетчик: «Conflicts/sec» показывает число зафиксированных в секунду конфликтов, на протяжении сеанса репликации слиянием. Счетчик: «Downloaded Changes/sec» показывает какое число строк в секунду копируется от издателя подписчику, и наоборот, число строк, копируемых в секунду от подписчика издателю показывает счетчик производительности: «Uploaded Changes/sec».

С помощью трассировки запросов репликации, собранной программой SQL Profiler, можно определить, кто является виновником повышенной нагрузки или задержек в тиражировании данных. Программа SQL Profiler отображает запросы, посылаемые клиентами на компьютер SQL Server, и обеспечивает информацию о каждом переданном серверу запросе. Главным событием для анализа является RPC: Completed из класса событий хранимых процедур, которое позволяет увидеть какие запросы и операции агенты исполняют на SQL Server. Полезную информацию будут содержать столбцы TextData и Duration. Если потребуется, дополнительную полезную информацию можно будет найти в столбцах DatabaseId и Spid (Server Process ID), которые позволяют различать подключения издателя и распространителя, если они находятся на одном сервере. Информация в столбцах LoginName и HostName будет полезна для идентификации подписчиков. Также, из-за очень большого количества анализируемых событий, полезно накладывать фильтры на собираемую в трассировке информацию, используя для этого системные имена агентов или таких программ, как SQL Server Enterprise Manager или SQL Profiler.

Если перечисленных событий и колонок недостаточно, можно рассмотреть необходимость добавления в трассировку следующих событий: SQL:StmtStarting, SQL:StmtCompleted, Scan:Started, Scan:Stopped, Show Plan Statistics, Show Plan All и Show Plan Text.

Задания обслуживания репликации

Кроме описанных выше основных агентов репликации и достаточно экзотического агента очередей, существуют еще несколько агентов репли-

кации, которые не представляют собой отдельные исполняемые модули, это простые задания агента сервера, расписанием которых можно управлять и этого, зачастую, бывает достаточно для настройки их работы. Основная задача большинства из этих агентов это автоматизация очистки устаревших метаданных и записей журналов хронологии работы агентов репликации, которые хранятся в базе данных распространителя. Давайте кратко рассмотрим их назначение.

Для очистки устаревшей хронологии работы агентов используется специальное задание: «Agent History Cleanup Agent». По умолчанию, в SQL Server 2000 это задание запускается каждые 10 минут, и удаляет записи старше 48 часов. Это помогает сократить размер системных метаданных.

Задание «Очистка распространения» (Distribution Cleanup Agent) запускается каждые 10 минут и удаляет успешно реплицированные всем подписчикам команды, метаданные которых хранятся в системных таблицах базы данных распространителя (значение по умолчанию). Кроме этого, удаляются все моментальные снимки, которые уже были применены подписчиками, и с момента применения последнего снимка прошло больше 72 часов (значение по умолчанию). Удаление снимков позволяет избежать исчерпания дискового пространства под моментальные снимки. Если к публикации разрешены анонимные подписки с опцией немедленного создания первоначального снимка, в папке снимков сохраняется хотя бы одна копия моментального снимка (не подлежащая очистке). Это гарантирует то, что для анонимных подписчиков всегда будет доступен самый «свежий» моментальный снимок. Подписки, которые не были синхронизированы в течение максимального срока хранения распространения, также деактивируются в процессе работы этого задания.

Очистка истекших подписок осуществляется заданием «Expired Subscription Cleanup Agent». По умолчанию, задание запускается каждый день в час ночи, и удаляет все просроченные подписки к издаваемой базе данных.

Задание «Повторная инициализация подписок, имеющих сбои при выполнении проверки данных» (Reinitialize Subscriptions Having Data Validation Failures Agent) призвано автоматизировать реакцию на неудачи при сверке данных издателя с подписчиком. По умолчанию, это задание остановлено и не имеет расписания. Сделано это потому, что подписчики, у которых выявлены расхождения данных с издателем, будут принудительно инициализированы и применят последний моментальный снимок при следующем запуске агента слияния или агента распространителя.

Для поиска агентов репликации, которые не подают «признаков жизни», используется специальное задание: «Replication Agents Checkup Agent». Это задание запускается каждые 10 минут, и делает записи в системных журналах, если обнаруженные агенты репликации, которые долгое время не делали хронологических записей о сеансах своей работы.

Есть еще один агент репликации, который не имеет своего собственного задания, работает постоянно и настраивается на одной из вкладок оснастки монитора репликации. Название этого агента: «Replication monitoring refresher for distribution», нужен он для настройки обновления наблюдаемой в мониторе репликации картины активности агентов. В его задачу входит обновление буферов запросов, используемых монитором репликации.

Подготовка к репликации

Подготовка системы к реализации репликации данных должна начинаться с этапа тщательного планирования. Вначале необходима адаптация баз данных к требованиям репликации, а также адаптация коммуникации серверов к порождаемой репликацией нагрузке. Планирование должно также включать вопросы организации информационной безопасности транспорта репликации и хранения данных, а также планов резервирования и восстановления репликации и данных. Очень полезно иметь подробное описание и план внедрения репликации и запуска ее в эксплуатацию.

Адаптации к репликации, в первую очередь, подлежит схема данных. Например, если для организации ссылочной целостности в базе данных используются поля IDENTITY, и приложение подразумевает ввод новых записей не только на издателе, но и на подписчике, нужно предусмотреть меры по разделению диапазонов IDENTITY между подписчиками и издателем или заменить столбцы IDENTITY на столбцы глобальных уникальных идентификаторов – GUID.

Если в публикацию в виде статей включаются не все таблицы базы данных, необходимо принять меры на уровне схемы, чтобы обеспечить возможность выполнения каскадных операций и поддержку ссылочной целостности между таблицами, входящими и не входящими в публикацию. Аналогичные меры нужно продумать для триггеров, поскольку срабатывание триггера на привнесенные репликацией операции может быть нежелательно, и для такого триггера нужно использовать параметр NOT FOR REPLICATION. Старайтесь создавать дополнительные триггеры, индексы и представления на издателе, а не на подписчиках.

Пользовательские приложения должны уметь работать с фильтрованными репликацией таблицами, это могут быть как горизонтальные фильтры, так и вертикальные фильтры.

При проектировании таблиц для репликации транзакций, в целях повышения эффективности работы репликации, стоит придерживаться нескольких простых рекомендаций. Во-первых, все таблицы, включенные в репликацию транзакций, должны иметь первичные ключи. Во-вторых, для входящих в публикацию таблиц стоит рассмотреть возможность отключения срабатывания триггеров и проверок ограничений внешнего ключа для реплицируемых операций. Ограничения внешнего ключа и триггеры

используются для обеспечения целостности данных. Поддержка ссылочной целостности в репликации может обходиться слишком дорого, поскольку любые изменения будут приводить к срабатыванию этих механизмов на всех подписчиках. Кроме того, такие срабатывания зачастую паразитные, поскольку дублируют уже выполненные на издателе срабатывания. Для того, чтобы отключить срабатывание триггеров и ограничений внешних ключей, Вы можете пересоздать их с опцией NOT FOR REPLICATION. Объекты, созданные с опцией NOT FOR REPLICATION будут срабатывать только в ответ на действия пользователей, но не на действия репликации. Опция таблицы NOT FOR REPLICATION становится активна только тогда, когда к таблице обращается агент репликации.

Кроме адаптации схемы данных и приложений необходимо учесть ту дополнительную нагрузку, которую репликации может создавать для серверов и коммуникационных сетей. Пропускная способность каналов передачи данных между узлами репликации является одним из основных факторов производительности тиражирования и синхронизации данных. Существенную дополнительную нагрузку репликация оказывает и на аппаратные ресурсы серверов в виде дополнительных подключений, параллельных операций, утилизации оперативной памяти под нужды агентов репликации, повышению требований к производительности дисковой подсистемы и сетевых интерфейсов.

Кроме технических вопросов, внедрение распределенной системы с репликацией может потребовать изменения бизнес-правил и требований ко всему информационно-техническому комплексу. Реализацию таких изменений нужно планировать заранее, тщательно макетировать и только потом приступать к внедрению и эксплуатации. Вносить изменения в распределенную систему намного сложнее, чем в систему автоматизации одного офиса.

Система, использующая репликацию, включает в себя довольно большой список оборудования и программного обеспечения, которое задействуется на разных участках обслуживания и транспорта реплицируемой информации. Даже в простой реализации, в этот список можно включить сервера и компьютеры, а также все активное и пассивное сетевое оборудование. При репликации через интернет этот список существенно возрастет за счет оборудования, обеспечивающего внешние коммуникации и безопасность сегментов сети.

Поскольку мы имеем дело со сложной системой, возрастает количество критически важных компонент, сбой которых приведет к потере производительности или краху репликации.

В цепочке репликации участвуют, как сложные элементы, так и простые, на которые в обычных условиях не обращают пристального внимания. Но поскольку в нашем случае они также становятся важны, необходимо контролировать работу всех компонентов распределенной системы.

Обслуживание системы с репликацией требует более дорогого администрирования и более сложного разрешения проблем. Это связано в

первую очередь с резким увеличением сложности распределенной системы, которая требует больших временных затрат и высокой квалификации администратора и программистов баз данных. Кроме того, тут не обойтись без привлечения специалистов из смежных областей, таких как сетевые администраторы, администраторы безопасности и системные администраторы.

Важным аспектом подготовки репликации является мониторинг всех компонент, задействованных в цепочках передачи тиражируемых данных. Мониторы репликации, поставляемые с разными версиями SQL Server, помогают идентифицировать множество различных проблем репликации, но в рамках одного инструмента невозможно сконцентрировать возможности наблюдения за всем многообразием аппаратных и программных средств, которые могут использоваться в топологии репликации. Еще на стадии планирования репликации важно четко представлять, какие дополнительные средства мониторинга и диагностики необходимо будет мобилизовать для обеспечения бесперебойной и надежной работы всех используемых репликацией компонент распределенной системы. Те же компоненты, работоспособность которых зависит от сторонних фирм, должны иметь четкие и понятные аварийные планы, с помощью которых можно было бы максимально быстро и эффективно обнаружить и разрешить проблему.

Наиболее распространенной сетевой средой локальных вычислительных сетей сегодня является Ethernet. Практически повсеместно используются сети пропускной способностью 1000 или 100 Мб/сек. Необходимо помнить и об ограничениях, которые накладывает такая сеть.

Сети Ethernet являются коллизионными, т.е. возникающие одновременно в сети два пакета отклоняются оборудованием сети и повторяются через случайный промежуток времени, разный для каждого из пакетов. Наличие большого числа коллизий может привести к снижению производительности операций, которые сервер баз данных осуществляет через сеть.

Сети Ethernet имеют ограничение на протяженность сегмента. Превышение длинны сегмента может привести к тому, что ответ о получении пакета не будет вовремя получен узлом, его ожидающим. Такая ситуация приводит к потерям пакетов, что тоже негативно отражается на производительности.

Сети Ethernet имеют ограничение на число соединений между парой устройств активного сетевого оборудования. Превышение установленного стандартом числа пассивных соединений может увеличить время отклика и породить неоднородности или аномалии в сети, снижающие ее производительность.

Протяженность сети, наличие между серверами большого количества активного и пассивного сетевого оборудования, имеющего собственную задержку обработки запроса, также может снижать производительность, хотя будут соблюдены требования стандарта, и каждый участок в отдельности будет работать с высокой производительностью.

Сегодня многие локальные сети связаны между собой протяженными коммуникационными линиями, имеющими меньшую пропускную способность, чем ЛВС. Для этого применяются модемные соединения через обычные телефонные линии или выделенные линии, каналы T1 и E1, фрейм-релейные сети, радио Ethernet, разного рода беспроводные технологии передачи данных, широкополосный интернет и т.д.

По существу, все такие соединения накладывают ограничения на пропускную способность коммуникаций между серверами и имеют высокие значения задержки передачи запросов. Используемое на коммуникационном канале оборудование, контролирующее заявленную полосу пропускания, при превышении трафика может «резать» превышающие трафик пакеты, что также может приводить к снижению пропускной способности канала в целом или даже к ошибкам в сеансе синхронизации данных.

Нужно также учитывать, что провайдеры глобальных сетей не всегда могут гарантировать заявленную полосу пропускания, возможна перегрузка канала на отдельных участках глобальной сети, что может отразиться на трафике репликации.

Плохо спланированная локальная сеть или недостаточная пропускная способность интернет-канала также могут создать проблемы при организации репликации. Для своевременной реакции на возникновение проблем репликации стоит наладить постоянный контроль утилизации сетевых ресурсов и обеспечить возможность диагностики всей топологии вашего внешнего и внутреннего коммуникационного канала для репликации. Размещайте сервера в центральных сегментах и старайтесь обеспечить высокую производительность обслуживающих репликацию сегментов. Обеспечьте максимально возможную пропускную способность для внешнего коммуникационного канала, используемого для репликации. Лучше иметь для этого отдельный канал, не используемый для других целей. Будьте осторожны в применении диагностических и журналирующих сетевой трафик систем. Они способны создавать собственный трафик в сети, конкурирующий с трафиком репликации. При этом, увеличение числа сетевых ошибок будет способствовать росту служебного трафика и трафика репликации, что может привести к затоплению всего канала.

В репликации слиянием, начиная с SQL Server 2000, присутствуют встроенные средства регулировки ресурсов. Затопление внешнего коммуникационного канала или чрезмерную утилизацию аппаратных ресурсов серверов издателя и подписчиков можно предотвратить за счет ограничения числа одновременно допустимых подключений, создаваемых агентами репликации слияния подписчиков. При этом следует учитывать это ограничение при разработке расписания работы агентов слияния. Внести соответствующее ограничение в SQL Server 2000 можно на вкладке Subscription окна свойств публикации.

Появление коммуникационных проблем, особенно при репликации через интернет, чревато лавинообразным ростом трафика репликации. Например, значительный посторонний трафик может привести к суще-

ственному увеличению продолжительности сеансов репликации. Задания на запуск агентов будут отрабатывать долго, и практически сразу запускаться снова. Возрастет количество сбоев в передаче данных и повторов попыток агента провести сеанс синхронизации данных. Увеличится доля служебного трафика, и так далее, подобно эффекту снежного кома... В таких случаях, самым простым и действенным средством на время локализации коммуникационных проблем является изменение расписания запуска сеансов репликации. Разнесение по времени сеансов разных подписчиков также позволит сократить взаимное влияния их трафика друг на друга. Для предотвращения приводящего к затоплению сети поведения агентов репликации можно отказаться от повторных попыток синхронизации при неудаче. После этого можно автоматизировать изменение периодичности запуска агента репликации и обеспечить принудительное прекращение сеанса репликации, если его продолжительность превысила допустимое время.

Еще одной опасностью для репликации через сети передачи данных является запуск процедур сверки данных между издателем и подписчиками. Это длительная по своей природе операция, которая сопровождается значительным сетевым трафиком, причина возникновения которого связана с необходимостью выполнения распределенных запросов для сверки данных по каждой статье публикации и, возможно, для каждой пары издатель/подписчик. Необходимость в сверке данных издателя с подписчиком может возникнуть тогда, когда вы подозреваете, что не все записи в процессе репликации были переданы или, например, записи на подписчике могли быть удалены по неосторожности.

Сверка может выполняться со всеми, либо только с одним подписчиком, это указывается для SQL Server 2000 в специальном диалоговом окне программы Enterprise Manager, которое называется: «Validation Options». Существует три вида сверки, которые отличаются степенью нагрузки на серверы и каналы связи.

Можно выбрать быстрый подсчет строк (самый щадящий метод), точный подсчет строк (для сверки с другими версиями) и самый «тяжелый» способ – вычисление контрольных сумм. Сверка может выполняться не только вручную, но и автоматически, по расписанию. Кроме того, если сверка выявит расхождения, можно настроить автоматическую повторную инициализацию для не прошедшего сверку подписчика. Для этого в SQL Server 2000 существовала опция: «Reinitialize Subscriptions Having Data Validation Failures Agent».

В состав системных хранимых процедур входят три специализированные процедуры, которые предназначены для реализации сверки данных. Хранимая процедура sp_article_validation предназначена для сверки одной статьи и применима в репликации транзакций (для некоторых версий она применима и для репликации снимков). Процедура sp_publication_validation сверяет всю публикацию репликации транзакций (для некоторых версий она применима и для репликации снимков).

И еще одна процедура применима для всех типов репликации и позволяет осуществлять сверку таблиц или индексированных представлений. Называется эта системная хранимая процедура: sp_table_validation. Ее нельзя применять для таблиц, публикуемых издателем Oracle.

Развитие репликации

Репликация не стоит на месте, она постоянно развивается, и новшества появляются не только в новых версиях, но и буквально в каждом сервисном пакете. Наиболее значимый шаг был сделан еще в SQL Server 7.0. В следующей версии SQL Server 2000 к четвертому сервисному пакету репликация заработала очень стабильно и надежно.

В SQL Server 2005 появилась целая серия новинок, одной из ожидаемых была репликация DDL (команд языка определения данных: ALTER TABLE, ALTER VIEW, ALTER PROCEDURE, ALTER FUNCTION и ALTER TRIGGER). Для предыдущих версий SQL Server поддерживал только добавление или удаление столбца статьи. Это нововведение позволяет существенно упростить изменения схемы публикуемой базы данных.

Теперь изменения схемы можно выполнять с помощью программы SQL Server Management Studio, запуская в окне запроса инструкцию DDL из состава языковых конструкций Transact-SQL, или можно использовать объекты SQL Management Objects (SMO) в собственных приложениях. DDL команды, по умолчанию, будут растиражированы всем подписчикам в сеансах агента распространителя или агента слияния.

Изменения схемы можно делать только на издателе, и оно подчиняется всем ограничениям для команды ALTER, предписываемым правилами языка Transact-SQL (например, нельзя изменять поля первичного ключа). Есть и целый ряд других ограничений. Репликация слиянием не позволяет изменять схему на подписчике, а в репликации транзакций это возможно, но может привести к сбоям. Также, в репликации слиянием нельзя удалять поля со свойством uniqueidentifier или ROWGUIDCOL. Изменения схемы, переданные переиздающему подписчику, по умолчанию будут тиражироваться и на его подписчиков. Если изменения схемы затрагивают объекты, имеющие ссылки или ограничения, связанные с объектами, которые есть на издателе, но их нет на подписчике, изменение пройдет успешно на издателе, но окончится неудачей на подписчиках.

Все используемые в инструкции DDL объекты подписчика должны иметь одинаковые с издателем имена и одних и тех же владельцев. Все ограничения должны иметь имена, иначе они могут быть именованы поразному (автоматическое именование) на издателе и подписчиках. В инструкциях DDL необходимо использовать трехсоставное именование, только если база данных подписчика имеет такое же имя как на издателе.

Изменения схемы, тиражируемые подписчикам других СУБД, требуют повторной инициализации. Не поддерживаются изменения схемы

для издателей других типов СУБД. Также, повторная инициализация потребуется в репликации транзакций после добавления колонки со свойством timestamp.

В SQL Server 2000 вы не могли для изменения набора столбцов статьи использовать команду ALTER TABLE, для этого нужно было использовать системные процедуры sp_repladdcolumn и sp_repldropcolumn. Кроме этого, существовала возможность внести изменения в окне свойств публикации или использовать соответствующие объекты SQL Distributed Management Objects (SQL-DMO).

Еще один способ исполнения сценария DDL на всех подписчиках предоставляет системная хранимая процедура sp_addscriptexec. Ее использование немного отличается в разных версиях, поэтому перед ее использованием ознакомьтесь с описанием ее работы в электронной документации по SQL Server.

Репликация в SQL Server 2005 может работать более производительно, чем в прежних версиях, с помощью использования возможности динамической фильтрации. Это возможно за счет введения используемых по умолчанию предварительно вычисляемых секций (Precomputed Partitions). Эта новая опция стала теперь доступна в свойствах публикации. Предварительно вычисленные по возможным значениям динамических фильтров секции позволяют без задержек на фильтрацию начинать сеанс синхронизации подписчика. Для этого необходимо выполнение нескольких условий. Любые функции, используемые в динамических фильтрах, такие как HOST_NAME() и SUSER_SNAME(), должны использоваться только в условии фильтрации и не могут находиться внутри представления, join-фильтра или пользовательской динамической функции. Представления, используемые в join-фильтрах, тоже не должны содержать динамические функции. Значение, возвращаемое для каждого подписчика, не должно измениться после того, как раздел был создан. Например, для HOST_NAME() нельзя будет изменить имя сервера подписчика. Условия динамической фильтрации и join-фильтры не могут использовать поля text, ntext или image. И последнее, в публикации с join-фильтром не должно быть замкнутых связей.

Кроме больших нововведений, в SQL Server 2005 появились и не столь значимые, но зато добавляющие удобства в работе функции. Так, например, теперь стала возможной инициализация подписчика репликации транзакций из резервной копии. Повышена производительность синхронизации данных во время сеанса репликации. Более удобным стал монитор репликации. Добавлены предупреждения о проблемах производительности. Можно измерить задержки репликации транзакций по завершению подписчиком начатой на издателе транзакции. Реализовано добавление в сеанс репликации транзакций записей трассировки производительности. Статистика сеанса репликации слиянием углублена до уровня статьи. Можно видеть процент исполнения сеанса агентом репликации слияния. Добавлена процедура, позволяющая корректно менять

метаданные репликации после смены имени сервера. Для этого нужно выполнить следующую инструкцию:

```
EXEC sp_rename_replication_server 'SubscriberA', 'SubscriberB'
```

Новые, максимальные типы данных для больших значений не имеют присущих текстовым значениям недостатков и ограничений репликации. Такие типы, как varchar(max), nvarchar(max) и varbinary(max) можно применять вместо привычных типов: text, ntext и image.

В SQL Server 2008 получила дальнейшее развитие одноранговая репликация транзакций. Появилась возможность автоматического обнаружения конфликтов сеансов репликации, которые отслеживает агент распространителя и это поведение включено по умолчанию. Также, в этой версии добавление узлов в топологию репликации не требует остановки репликации по всей топологии до того момента, пока изменения не будут доставлены всем узлам. Для этого был добавлен еще один тип синхронизации "init from lsn", который передается в качестве параметра @sync_type системной хранимой процедуры sp_addsubscription. Разумеется, все это возможно сделать и с помощью соответствующего мастера определения топологии одноранговой репликации транзакций. Кстати, этот входящий в состав SQL Server 2008 мастер позволяет управлять топологией одноранговой репликации с помощью визуальных средств.

Монитор репликации в SQL Server 2008 получил усовершенствования пользовательского интерфейса. Выводимые на экран в виде таблиц метаданные сеансов репликации теперь можно сортировать по нескольким столбцам, накладывать вертикальные и горизонтальные фильтры. Кроме того, внесены некоторые косметические изменения в окна мастеров и свойств объектов репликации.

Заключение

В этой статье был дан краткий обзор репликации и нюансов работы ее агентов в SQL Server. Вам был представлен не типовой обзор репликации, а взгляд со стороны агентов репликации и представлены основные особенности их работы. Такой подход ближе администраторам баз данных. Далее, в этой книге, вы найдете множество разных подходов и рецептов, пригодных не только для администраторов, но и для разработчиков приложений баз данных. Книга представляет собой сборник совершенно самостоятельных статей, которые собраны в разделы, посвященные разным типам репликации, и сгруппированы по двум основным версиям: SQL Server 2005 и SQL Server 2008. Такая структура книги поможет вам использовать ее не только для изучения особенностей использования и настройки репликации, но и для решения конкретных прикладных задач.

Надеюсь, собранный в этой книге опыт окажется для вас полезным, а сама книга будет востребована и станет настольной и для администраторов баз данных и для разработчиков приложений

Общие вопросы

Резервное копирование в репликации SQL Server 2008

Автор Александр Юрьевич Гладченко

В этой статье рассматриваются требования к резервному копированию баз данных, задействованных в репликации моментальных снимков, репликации транзакций или репликации слиянием. Эти требования зависят от роли, которую сервер исполняет в репликации и от места, где в топологии репликации необходимо обеспечить восстановление тиражируемых данных. Чтобы восстановить участвующие в репликации данные, необходимо регулярно резервировать издателей, дистрибуторов и подписчиков. Статья основана на материалах электронной документации, поставляемой в дистрибутиве сервера баз данных SQL Server 2008 Books Online.

Стратегии резервирования

Правильный выбор стратегии резервирования баз данных, участвующих в репликации, позволит максимально сократить время восстановления работоспособности распределенной системы после отказа или аппаратного сбоя. Стратегия резервирования зависит от требований к времени восстановления и от его сложности, обычно описываемой в плане резервирования. Как правило, если нужно быстро восстановить любую резервную копию при минимальной вероятности потери данных, обеспечивается полное резервное копирование всех пользовательских и системных базы данных.

Можно выделить четыре наиболее распространенных стратегии резервирования, используемых в репликации, которые представлены ниже и определяются списком копируемых баз данных.

- Публикуемые базы, master и msdb.
- Публикуемые базы, база распределителя, master и msdb.
- Публикуемые базы, подписываемые базы, master и msdb.
- Публикуемые базы, подписываемые базы, база распределителя, master и msdb.

Эти стратегии могут быть расширены за счет вовлечения в резервирование системных баз данных подписчиков или дистрибутора, если он

работает на выделенном сервере. Создание полных резервных копий публикуемых баз данных на издателе и использование имеющейся в репликации SQL Server возможности повторной инициализации одной или нескольких подписок, является наиболее простой стратегией резервирования. Эта стратегия может использоваться при большом числе обычных или мобильных подписчиков, администрирование баз данных которых затруднено или невозможно. Для полного восстановления репликации, достаточно будет иметь полную резервную копию публикуемых баз и набор реализующих репликацию сценариев. Главным недостатком этой стратегии является то, что в случае отказа дистрибутора или издателя придется заново устанавливать репликацию.

Более надежной (но все еще простой) стратегией является резервирование издателя и дистрибутора в тот момент, когда они синхронизированы. Эта стратегия также позволяет полностью восстанавливать репликацию. Резервное копирование подписчиков является не обязательным, но оно может уменьшить время возобновления работы подписчиков. Если ваша бизнес - логика приложений требует немедленного восстановления репликации, можно рассмотреть более сложные схемы резервирования и стратегии восстановления, которые будут описаны ниже в этой статье.

В большинстве сценариев резервирования, публикуемые базы данных и дистрибутор должны резервироваться сразу после добавления или изменения объектов репликации, например, статей и подписок или после того, как вносятся изменения в схему, затрагивающие репликацию. Если база данных дистрибутора будет восстановлена на тот момент времени, когда такие изменения еще небыли внесены, необходимо восстановить публикуемую базу данных на этот же момент времени. Из этого можно сделать вывод, что время серверов, участвующих в репликации должно автоматически синхронизироваться. Для синхронизации времени серверов можно использовать возможности службы времени операционных систем Windows 2000/2003/2008 или мультисерверное администрирование SQL Server 2008.

Как часть любой стратегии резервирования, всегда сохраняйте текущие сценарии создания объектов репликации и параметры ее настройки (профили, параметры запуска агентов, сценарии заданий запускающих агентов и т.п.) в надежном и безопасном месте. Также очень полезно иметь сценарии создания и удаления репликации.

Еще в SQL Server 2000 появилась возможность восстановления издаваемых или подписанных баз данных на другом сервере или на том же сервере, но с другим именем. Для этого в команду RESTORE нужно добавить опцию KEEP_REPLICATION. По умолчанию, при восстановлении издаваемой или подписанной в репликации слиянием базы данных на другом сервере вся информация о репликации будет утеряна. Если же нужно наоборот, удалить информацию о репликации после восстановления на том же сервере, запустите системную хранимую процедуру

sp_restoredbreplication, которая удалит все относящиеся к репликации метаданные.

Резервное копирование издателя

Публикуемые базы данных являются наиболее важным и уязвимым звеном топологии репликации, поэтому, даже самый простой план резервирования должен включить полное резервное копирование издателей. Резервное копирование издателя должно быть регулярным и кроме полной копии издаваемой базы должно включать резервирование журнала транзакций и/или дифференциальное резервирование. Также можете копировать базы master и msdb, чтобы обезопасится от полной потери системы, а не только пользовательских баз данных. Если используется доставка журналов (Log Shipping), обязательно регулярно копируете системную базу данных msdb (которая используется для учета передачи копий журнала). Также, нужно делать резервные копии после внесения любых изменений в публикации или выполнения процедур по обслуживанию публикуемой базы данных.

Перед тем, как приступить к восстановлению издателя после полного системного краха или потери издаваемых баз данных, нужно с помощью системной хранимой процедуры sp_replicationdboption отметить восстанавливаемые базы данных, как участвующие в репликации соответствующего типа. Если этого не сделать, то информация о принадлежности к репликации не будет восстановлена.

Резервное копирование распределителя

Резервное копирование распределителя подразумевает резервное копирование базы данных distribution и системных баз данных msdb и master. Это позволяет восстановить его работоспособность практически после любого его отказа, без необходимости пересоздания публикаций или реконфигурации репликации.

Резервное копирование распределителя позволяет хранить данные о моментальных снимках публикаций, о хронологии сеансов репликации, об ошибках и другую информацию агентов репликации. Имея такую копию, можно ускорить восстановление издателя и распределителя, поскольку не потребуется переустанавливать репликацию. Особенностью в репликации транзакций является то, что ее стратегия резервирования требует координации между резервным копированием баз данных публикаций и распределителей. SQL Server 2008 умеет выполнять такую координацию автоматически. Резервируйте базу distribution, а затем делайте резервные копии журнала транзакций и дифференциальные резервные копии этой базы данных. Восстанавливать базу данных распределителя нужно синхронно с восстановлением издаваемой базой данных. После восстановления базы распределителя стоит проверить пара-

метры профилей используемых агентов репликации, чтобы убедиться в их соответствии бизнес - требованиям приложения. На одном сервере, исполняющем роль распределителя, может находиться несколько баз данных распределения для разных издателей. В такой конфигурации, важно следить за координацией резервирования издателей и базы дистрибутора, что бы за счет этого сократить время на повторную инициализацию подписчиков.

Резервное копирование подписчика

Простая стратегия резервирования может основываться на повторной инициализации подписок вместо восстановления их из резервной копии, и может быть расширена за счет добавления в план резервирования создания полных резервных копий каждой подписанной базы данных и системных баз подписчика msdb и master. Базы данных msdb и master необходимо копировать, если используется pull - подписка и это потребуется только при восстановлении после потери всей системы.

Нужно делать полную резервную копию подписанной базы данных, а затем делать резервные копии журнала транзакций и дифференциальные копии базы.

Обратите внимание на то, что резервное копирование каждого подписчика не является обязанным для восстановления репликации после отказа. В большинстве случаев, регулярное резервирование издателя и дистрибутора достаточно для решения этой задачи. Если затраты на повторную инициализацию подписчика значительно больше чем на его восстановление из резервной копии, и сложность управления резервным копирование в масштабе всего предприятия не велика, Вы можете рассмотреть возможность расширения схемы резервирования резервным копированием подписчиков. Кроме того, если в репликации задействовано только несколько статей и на подписчике существуют таблицы, которые не могут быть повторно инициализированы, Вам также потребуется резервировать базы данных на подписчиках. Понадобится это и при переиздании участвующих в репликации статей.

Резервирование баз данных master model, msdb и distribution

Базы данных master, model, msdb и distribution резервируются так же, как и пользовательские базы, и для них (если имеются изменения) эта операция тоже должна выполняться регулярно. Во избежание конкуренции с системными объектами, не рекомендуется создавать в этих базах пользовательские объекты. Также, следует быть осторожным при использовании утилиты восстановления системных баз данных rebuildm.exe, т.к. в процессе ее работы все системные базы данных заме-

няются шаблонами из дистрибутива и хранимая в системных базах информация будет утеряна. При выборе стратегии резервирования, нужно учитывать тот факт, что базы master, model, msdb и distribution поддерживают все существующие модели резервирования, от полной до простой.

База данных master издателя хранит в таблице sysservers информацию о подписчиках. В такой же таблице базы данных master на дистрибуторе хранится информация об издателях. Для обеспечения возможности восстановления репликации после потери издателя или дистрибутора, делайте резервные копии их баз master после изменений числа подписчиков или издателей. Наиболее важной таблицей базы данных master является sysdatabases, т.к. от того, сохранятся ли правильные значения в поле category для реплицируемых баз, зависит успешность их дальнейшего восстановления без потери настроек публикаций. Для репликации транзакций или моментальных снимков это поле должно содержать единицу, а для репликации слиянием - четверку. При восстановлении базы данных на тот же сервер, где была выполнена ее копия, SQL Server анализирует установленное для этой базы в поле master.dbo.sysdatabases.category значение, по которому ему становится ясно, нужно ли сохранять настройки репликации при восстановлении. Для того, что бы явно указать, что восстанавливаемая база данных является публикуемой в репликации транзакций или слиянием, нужно использовать системную хранимую процедуру sp_replicationdboption.

База данных msdb используется службой SQL Server Agent, программой Enterprise Manager и службой SQL Server для хранения информации о заданиях и их расписаниях, о резервных копиях и их хронологии. Информация в этой базе изменяется при планировании заданий, при сохранении DTS пакетов, при резервировании и восстановлении, а также во время сеансов репликации. SQL Server автоматически сохраняет данные о выполненном резервном копировании и хронологию восстановления данных в базе msdb. Запоминается, кто выполнил резервирование, в какое время и в каких устройствах или файлах хранится резервная копия. Именно эту информацию использует программа Enterprise Manager для вычисления плана восстановления базы данных с учетом необходимой для правильного восстановления хронологии восстановления копий журнала транзакций и дифференциальных копий. Операция резервного копирования для любых баз данных запоминаются, даже если они были выполнены из других приложений, не поставляемых в дистрибутиве сервера баз данных (например, использующих для этого SMO). Если Вы используете для резервирования и/или восстановления информацию о хронологии из базы msdb, рекомендуется установить для нее полную модель резервирования (Full Recovery model). В дополнение к этому, стоит разместить журнал транзакций базы msdb на устойчивой к сбоям дисковой подсистеме.

Если база данных msdb будет повреждена, и не будет актуальной резервной копии, то вся информация планировщика службы SQL Server Agent будет потеряна и ее придется восстанавливать вручную. Также будет утеряна информация о хронологии резервирования.

База данных distribution используется такими компонентами репликации служба SQL Server, как агент дистрибуции (Distribution Agent), агенты чтения журнала транзакций (Log Reader Agents), агент создания моментальных снимков (Snapshot Agent) и агент репликации слиянием (Merge Agent). Она хранит информацию о транзакциях, моментальных снимках, о состоянии синхронизации, и о хронологии репликации. Такая база данных располагается на сервере, который исполняет роль распределителя или совмещает эту роль с ролью издателя. Меньше всего от базы данных distribution зависит репликация слиянием (merge).

Если база данных distribution будет повреждена, и не будет актуальной резервной копии, то вся информация о репликации, используемая утилитами репликации SQL Server, будет потеряна и ее придется восстанавливать вручную. По этой причине, рекомендуется установить для базы distribution полную модель резервирования (Full Recovery) и расположить ее журнал транзакций на устойчивой к сбоям дисковой подсистеме.

База данных model является шаблоном, используемым SQL Server при создании других баз данных, например, tempdb или пользовательских баз. После создания базы данных, все объекты базы данных model, включая ее настройки, будут скопированы в новую базу данных. База данных model непосредственно не задействуется в репликации, но нужна при создании новых баз с преопределенными в ней настройками. Поэтому, резервную копию этой базы достаточно создавать при внесении в нее изменений.

Если база данных model будет повреждена, и не окажется ее актуальной резервной копии, все изменения шаблона создания пользовательских баз данных будут потеряны и их придется восстанавливать вручную.

Стратегии для репликации моментальных снимков

Стратегии резервирования/восстановления в репликации моментальных снимков являются самыми легкими для реализации и сопровождения. К тому же, поскольку этот тип репликации создает моментальные снимки публикаций, которые содержат не только данные, но и схему, нет необходимости резервировать публикуемую базу так же часто, как это делается в репликации транзакций или репликации слиянием. Хорошей практикой является резервное копирование публикуемой базы данных сразу после внесения изменений в настройки или схему существующих публикаций или после добавления новых.

Вместе с издателем нужно резервировать и распределителя, что бы при необходимости восстановления они были синхронны. Во время их резервирования нельзя создавать новые моментальные снимки, публикации или добавлять подписчиков. Перед резервным копированием распределителя, полезно запустить на исполнение автоматически создаваемую мастерами репликации задачу Distribution Cleanup, после чего из базы данных дистрибутора будет удалена уже ненужная информация. Это может позволить сократить время генерации снимков и создания резервных копий базы distribution.

Во всем остальном, стратегия резервирования участвующих в репликации моментальных снимков баз данных не отличается от стратегии резервирования обычных пользовательских баз.

Стратегии для репликации транзакций

Начиная с версий SQL Server 2000 существует возможность восстановления издаваемых баз и базы распределителя, без необходимости повторной инициализации подписчиков или отключения/реконфигурации публикаций и дистрибутора. После восстановления, SQL Server автоматически скоординирует эти базы данных. Еще одним нововведением в SQL Server 2000 была возможность совместного использования репликации транзакций и службы доставки журналов на резервный сервер (log shipping).

Для обеспечения возможности восстановления издателя и дистрибутора на любой, заданный момент времени, необходимо для этих баз данных SQL Server включить опцию синхронизации реплицируемой базы данных с резервной копией. Делается это с помощью системной хранимой процедуры sp_replicationdboption, синтаксис вызова которой следующий:

```
EXEC sp_replicationdboption '<имя базы данных>', 'sync with backup', 'true'
```

При этом, Log Reader Agent, даже работая в непрерывном режиме, будет предоставлять распределителю для тиражирования только те транзакции, которые уже были скопированы из журнала транзакций в резервную копию. Т.о. нужно учитывать этот факт при задании расписания запуска резервного копирования журнала транзакций, чтобы оно выполнялось достаточно часто, в соответствии с бизнес - требованиями приложения ко времени тиражирования транзакций распределителем на сервера подписчиков. Этот механизм позволяет иметь в резервной копии издаваемой базы данных все транзакции, необходимые для синхронного с распределителем восстановления баз данных. Проверить, включена ли опция синхронизации с резервной копией можно с помощью системной функции DATABASEPROPERTYEX, например так:

```
SELECT DATABASEPROPERTYEX('<имя базы данных>', 'IsSyncWithBackup')
```

Из логики работы описываемого механизма следует, что издаваемые базы данных и база распределителя должны иметь полную модель резервирования, т.е. нужно отключить опцию «trunc. log on chkpt.».

В случае если Вы не включаете опцию синхронизации с резервной копией или не можете это сделать по каким либо причинам, восстановление издаваемых баз и базы дистрибутора возможно, но без повторной инициализации подписчиков может быть нарушена целостность тиражируемых данных, а данные издателя и подписчиков могут оказаться не синхронными. После восстановления издаваемой базы из резервной копии, запуск Log Reader - агента будет сопровождаться ошибкой, вызванной несогласованным состоянием с базой данных распространителя. Вызвав без параметров системную процедуру sp_replrestart, можно принудительно возобновить репликацию, а за счет добавления параметра запуска агента распределения (Distribution Agent) -SkipError, можно добиться работы этого агента с игнорированием ошибок, вызванных потерей синхронности с подписчиками. Восстановление базы данных на подписчике тоже может быть выполнено без необходимости его повторной инициализации. Для этого, нужно установить для дистрибутора равный, по умолчанию, нулю период хранения тиражируемых транзакций (minimum transaction retention period) на такой срок, что бы он был заведомо больше времени между резервными копиями журнала транзакций подписанной базы.

Каждая подписанная база данных содержат таблицу MSreplication_subscriptions, в которой хранятся данные о последних тиражированных подписчику транзакциях. Информация из этой таблицы используется для синхронизации версий реплицируемых объектов. После восстановления базы данных подписчика на том же сервере, версия которого отличается от сохраненной в резервной копии, для синхронизации версий объектов нужно выполнить системную хранимую процедуру sp_vupgrade_subscription_tables.

Стратегии репликации слиянием

Базы данных, участвующие в репликации слиянием, тоже могут быть восстановлены без необходимости последующей инициализации подписчиков и настройки публикации и подписок. Для этого используются последние данные, имеющихся на других серверах, которые можно синхронизировать с изменениями, не сохраненными в последней резервной копии. Этот тип репликации также можно совмещать со службой синхронизации журнала транзакций и резервного сервера (Log Shipping).

В репликации слиянием, все изменения данных фиксируются только в системных таблицах метаданных в публикуемой базе и в базах подписчиков, вследствие чего, отпадает необходимость в согласованности данных издателя и дистрибутора. Таблицы метаданных попадают в резервную копию так же, как и другие таблицы издаваемой или подписанной базы и поэтому отражаемое метаданными состояние синхронизации

подписчиков и издателя будет восстановлено в том виде, в каком оно попало в резервную копию. При этом гарантируется конвергенция данных во всех резервных копиях участвующих в репликации баз.

После восстановления издаваемой базы данных, можно просто повторно инициализировать всех подписчиков, что позволит привести их данные в соответствии с новым состоянием издателя. Однако, до повторной инициализации, можно синхронизировать издателя с теми подписчиками, у которых имеются последние данные, отсутствующие в резервной копии издателя. Т.о. потери данных не будет. Такую предварительную загрузку недостающих изменений можно сделать только с тех подписчиков, которые не являются анонимными и имеют полный комплект метаданных. Метод повторной инициализации применим и в том случае, если восстановление издателя выполняется на заданный момент времени, предшествующий выполненным в базе ошибочным действиям. После восстановления правильного состояния, создания моментального снимка и повторной инициализации всех подписчиков, все участвующие в репликации базы данные будут тоже приведены к правильному состоянию.

Резервную копию базы подписчика нужно создавать после успешной синхронизации с издателем. Если такая синхронизация, по каким-либо причинам, выполнялась давно, возрастает вероятность того, что относящиеся к актуальному на момент резервирования состоянию метаданные будут автоматически очищены (превышен retention period). В таком случае, синхронизировать данные подписчика с издателем без повторной инициализации будет невозможно. Если же резервная копия подписчика достаточно актуальна, то после восстановления по имеющимся метаданным подписчику будут переданы все изменения, которые происходили в реплицируемых данных после последней резервной копии подписчика. При этом повторная инициализация не потребуется. В случае переиздания подписанной базы или использования базы данных подписчика в качестве альтернативного партнера репликации, стратегия ее резервирования должна удовлетворять изложенным выше требованиям к издаваемой базе и базе подписчика.

Поскольку в репликации слиянием нет существенных отличий между базами данных издателя и подписчиков, требования по резервированию и восстановлению у них практически одинаковы.

Заключение

Реализация резервного копирования участвующих в репликации баз данных и серверов баз данных позволяет обеспечить полную и надежную защиту от потерь информации в результате сбоев программного обеспечения или аппаратных отказов. SQL Server 2008 предлагает для этого удобные и гибкие средства, которые хорошо знакомы администраторам баз данных, поскольку не выходят за рамки привычных программных мастеров и процедур управления и обслуживания сервера.

Шпаргалка по репликации в Microsoft SQL Server

По материалам статьи Робин Падж и Фил Фактор «SQL Server Replication Crib Sheet»

Перевод Яна Дмитриевича Либермана

Введение

Репликация – это набор технологий, позволяющих автоматически распространять данные из одной базы данных в одну или несколько других баз данных. Репликация используется для построения распределенных систем. Она также может быть полезна для построения систем с высоким уровнем доступности (high-availability systems). Репликация позволяет поддерживать данные в исходной и результирующих базах данных в согласованном состоянии. Она не подходит для единовременного копирования данных из одной базы данных в другую.

Типичными областями применения репликации являются следующие.

- Интеграция данных нескольких площадок (например, филиалов). Причем связь с удаленной площадкой может быть непостоянной.
- Интеграция данных из разнородных СУБД с использованием OLE DB провайдеров, интеграция данных мобильных пользователей, обмен данными с кассовыми терминалами.
- Хранилища данных и системы отчетности.
- Повышение уровня масштабируемости и доступности.
- Делегирование задач пакетной обработки данных.

Примеры использования репликации:

- Распространение данных относящихся к одному приложению другим приложениям, которые являются потребителями этих данных. Например, репликация данных продаж в систему подготовки отчетов.
- Обмен данными с мобильными пользователями, которые лишь время от времени подключаются к центральной базе данных для синхронизации.
- Создание нескольких экземпляров баз данных с целью распределения нагрузки между ними.

Microsoft SQL Server поддерживает три типа репликации: репликация моментальных снимков, репликация транзакций и репликация сли-

янием. Наличие различных типов репликации позволяет эффективно применять ее в очень широком спектре задач.

Репликация использует лексику издательской отрасли. Наименьшей единицей репликации является статья. Статьей может быть таблица, хранимая процедура или функция. Если это таблица, то к ней может быть применен фильтр для того чтобы реплицировать только нужные строки и столбцы. Все от статьи до базы данных целиком может быть реплицировано. Аналогия с издательскими терминами иногда может вводить в заблуждение. Так в репликации издатель, как правило, распространяет на подписчиков новые изменения, а подписчики в ряде сценариев могут вносить изменения в исходные данные.

Издатель содержит исходную копию данных. Издатель издает одну или несколько публикаций. Публикации состоят из статей.

Подписчики получают реплицированные данные от издателя. Подписчик может подписаться на одну или несколько публикаций. Любая база данных может одновременно выполнять функции издателя и подписчика.

Распространитель – это сервер, на котором находится база данных распространителя, хранящая метаданные и другие данные для всех типов репликации.

Репликация не является частью ядра SQL сервера. Она работает как внешнее приложение. Это облегчает поддержку других СУБД. Любые СУБД, для которых есть OLE DB провайдеры, могут быть издателями и подписчиками в репликации моментальных снимков и репликации транзакций.

Крайне важно правильно спланировать репликацию на начальном этапе. Особенно важно правильно выбрать тип репликации. Распространенная ошибка - использование репликации в случаях, где есть гораздо более простые решения.

Для производственных систем использующих репликацию, быстрое восстановление после сбоя может стать серьезной проблемой. Чтобы от этого защититься, нужно иметь хорошо проработанную и документированную стратегию восстановления, которая должна быть испытана и периодически проверяться. Это означает, что даже если репликация изначально была настроена с помощью графического интерфейса, должны быть созданы сценарии для всей топологии репликации, чтобы ее можно было вновь быстро воссоздать в чрезвычайных ситуациях. Использование графического интерфейса, делает первоначальное развертывания сравнительно легким, но это не самый лучший вариант, когда нужно быстро восстановить топологию репликации после сбоя.

Часто проблемы случаются, когда разработчики пытаются добавить или удалить какие-то статьи, изменить свойства публикаций, схемы опубликованных таблиц. Поэтому лучше начинать процесс настройки после того, как разработана относительно стабильная схема будущей репликации.

Топологии репликации

В большинстве случаев издателей, распространителей и подписчиков следует размещать на разных физических серверах.

Центральный издатель

Одна из самых распространенных схем репликации – это один издатель, хранящий исходные данные и один или несколько подписчиков. База данных распространителя может располагаться на одном физическом сервере с издателем или, что предпочтительней, на другом сервере.

Центральный подписчик

Если данные из нескольких баз данных, необходимо собрать в центральном хранилище, например, для системы отчетности, то настраивают одного подписчика подписанного на несколько публикаций.

Также, иногда можно встретить другие топологии репликации, такие как «двунаправленная репликация транзакций» и «одноранговая репликация транзакций». На самом деле они являются особыми вариантами топологий с центральным издателем или подписчиком.

Переиздающий подписчик

В ряде сценариев репликации, данные на подписчиков могут распространяться через других (переиздающих) подписчиков. Это позволяет репликации лучше работать на «слабых» каналах. То есть данные реплицируются по «слабому» каналу один раз на одного из подписчиков, а затем по быстрой внутренней сети распространяются на все остальные подписчики.

Методы репликации

Как работает репликация?

Репликация начинается с первоначальной синхронизации опубликованных объектов между издателем и подписчиком при помощи моментального снимка. Моментальный снимок содержит схему и данные всех объектов указанных в публикации. После того, как моментальный снимок создан на издателе, он доставляется подписчикам при помощи распространителя.

Для репликации моментальных снимков этого достаточно. Для других типов репликации, все последующие изменения в данных относящихся к публикации направляются подписчикам принудительно или по запросу.

Репликация моментальных снимков

Процесс создания и репликации моментального снимка (который содержит схему и данные всех заданных публикацией объектов) используется для первоначальной синхронизации подписок во всех типах репликации. Однако в ряде случаев эта первоначальная синхронизация и есть все, что необходимо. Например, когда данные меняются очень часто или их объем невелик или когда наличие последней версии данных не является необходимым для подписчика.

Процесс применения моментального снимка на подписчике включает копирование данных статей составляющих публикацию. Обычно, если те же данные на подписчике уже есть, то они переписываются (это поведение можно изменить). Репликация моментальных снимков обычно более затратная с точки зрения сетевого трафика, поэтому выполняется только через определенные интервалы времени. Блокировки, которые накладываются во время применения моментального снимка на подписчике, могут влиять на других пользователей. Поэтому этот тип репликации следует использовать в основном для редко изменяемых данных. В SQL Server 2005, несколько статей могут обрабатываться параллельно, а прерванное на середине применение моментального снимка может быть возобновлено с точки прерывания. Репликация моментальных снимков поддерживает немедленное обновление подписок и обновление подписок посредством очередей.

Изменения данных не отслеживаются для репликации моментальных снимков. Каждый раз, когда снимок применяется, он полностью перезаписывает имеющиеся данные.

Репликация транзакций

Репликацию транзакций следует использовать если:

- Изменения данных должны быть распространены на подписчиков как можно быстрее.
- Приложение, для которого создана подписка, должно обрабатывать каждое изменение данных.
- На издателе выполняется очень большое количество операций вставки, изменения и удаления.
- Издатель и подписчик – это разные СУБД, взаимодействующие посредством OLE DB.

После успешной синхронизации издателя и подписчика с помощью моментального снимка, все транзакции, зафиксированные на издателе, реплицируются на подписчика в том порядке, в котором они были сделаны. По своей сути, репликация транзакций передает данные только в одном направлении – от издателя к подписчикам. Однако поддерживается несколько сценариев допускающих внесение изменений в данные

на подписчиках. В зависимости от потребностей можно настроить обновляемую подписку с немедленным или отложенным (посредством очередей) обновлением.

Одноранговая репликация

Это особый тип репликации транзакций, в котором каждый участник является одновременно издателем и подписчиком (поддерживается только в Enterprise редакции). Одноранговая репликация наиболее полезна для групп, включающих до десяти серверов для балансировки нагрузки или для повышения уровня доступности.

Двунаправленная репликация транзакций

Под двунаправленной репликацией транзакций понимается схема, когда две базы данных реплицируют друг другу одну и ту же статью через распространителя. Для корректной работы этой схемы SQL сервер реализует механизм распознавания замыкания на себя. Конфликты, связанные с одновременным обновлением одних и тех же данных разными узлами в этой схеме не отслеживаются. Настраивать такую репликацию потребуется с помощью сценариев, так как сделать это с помощью графического интерфейса нельзя.

Репликация транзакций для отслеживания сделанных изменений использует информацию из журнала транзакций SQL сервера.

Репликация слиянием

Репликация слиянием позволяет нескольким узлам работать автономно, а затем объединить изменения в единый результат.

Репликация слиянием изначально задумывалась для поддержки мобильных и подключающихся время от времени подписчиков (пользователей). Также она подходит для построения систем с высоким уровнем доступности. Репликация слиянием предназначена для случаев, когда издатель и подписчики не поддерживают постоянную связь. После первоначальной синхронизации с помощью моментального снимка, последующие изменения отслеживаются на каждом узле локально с помощью триггеров. Затем, когда связь между издателем и подписчиком устанавливается, базы данных объединяются, используя ряд правил для урегулирования всевозможных конфликтов.

Репликация слиянием используется, когда несколько подписчиков могут менять одни и те же данные в разное время. Затем эти изменения реплицируются обратно на издатель, а затем к другим подписчикам. Она также подходит для приложений, в которых подписчики получают от издателя данные, переходят в автономный режим, вносят изменения, а затем соединяются с издателем для синхронизации изменений с издателем и другими подписчиками.

В репликации слиянием каждый подписчик имеет свою копию данных, в которую может вносить изменения. Когда данные объединяются,

могут возникнуть конфликты, связанные с отсутствием блокировок и, как следствие, возможностью изменения одних и тех же данных издателем и более чем одним подписчиком. Для идентификации и разрешения этих конфликтов используются правила, которые определяются на этапе настройки репликации.

В репликации слиянием не используются транзакции. Репликация слиянием обычно работает на уровне строк, но может работать и на уровне группы логически связанных строк. Также можно указать порядок, в котором статьи обрабатываются в ходе синхронизации.

Репликация слиянием отслеживает изменения, используя триггеры и таблицы метаданных.

Агенты репликации

Репликация реализована как набор агентов, которые являются отдельными приложениями, отвечающими только за какую-то определенную часть работы. В производственной среде агенты репликации не следует запускать от имени учетной записи SQL Server Agent. Напротив, нужно предоставить агентам только минимум прав, необходимый для их работы.

SQL Server Agent

Агенты репликации запускаются как задания в SQL Server Agent.

Агент моментальных снимков

Snapshot.exe выполняется на распространителе. Он извлекает схему и данные относящиеся к публикации, которые затем отправляется подписчикам через папку моментальных снимков. Он также обновляет статусную информацию в базе данных распространителя. Он используется во всех типах репликации.

Агент чтения журнала

LogRead.exe используется в репликации транзакций. Он читает нужные зафиксированные транзакций из журнала транзакции издателя, пакует и передает их распространителю в правильном порядке.

Агент распространителя

Distrib.exe принимает моментальные снимки и записи из журнала транзакций подготовленные агентами, о которых мы говорили выше, и направляет их подписчикам.

Агент слияния

ReplMer.exe используется только в репликации слиянием для применения к подписчикам исходного моментального снимка, а также для обмена изменениями между издателем и подписчиками.

Агент чтения очереди

QrDrSvc.exe используется в репликации транзакций, если для публикации включено отложенное обновление. Этот агент перемещает изменения, сделанные на подписчике, обратно на издатель.

Мониторинг репликации

Многие проблемы, связанные с репликацией можно избежать путем выполнения регулярных проверок. Лучшая проверка - это убедиться, что данные реплицируются, так, как это было задумано. Периодические проверки с помощью SQL Compare и SQL Data Compare, в дополнение к тому, что можно сделать стандартными средствами, могут оказаться очень полезными. Кроме того следует проверять процессы и задания репликации, что бы убедиться что они исправно работают.

Проверка производительности репликации

Производительность репликации следует регулярно контролировать и по мере необходимости вносить соответствующие изменения в настройки.

Монитор репликации используется для оперативного контроля состояния публикаций, а также инспектирования истории и ошибок. Его можно запустить с помощью контекстного меню узла «Репликация» в обозревателе объектов.

Одной из наиболее серьезных проблем являются задержки в распространении транзакций из базы данных издателя в базу данных подписчиков. В периоды, когда издатель сильно загружен, задержки могут быть значительными. Хранимая процедура sp_browseReplCmds возвращает набор команд репликации, хранимых в базе данных распространителя. С ее помощью можно определить, как сильно мы отстаем от синхронизации в любой конкретный момент времени. Начиная с SQL Server 2005, для того чтобы отслеживать реальные задержки в схеме репликации и соответственно идентифицировать «узкие» места, можно использовать трассировочные маркеры.

Проверка

Работает ли репликация, так как это было задумано? Для того, что бы можно было ответить на этот вопрос утвердительно, есть специальные хранимые процедуры, которые сравнивают статьи на издателе и подписчиках, что бы убедиться, что они одинаковые.

Хранимая процедура sp_publication_validation проверяет данные, каждой статьи вызывая хранимую процедуру sp_article_validation. Хранимая процедура sp_article_validation в свою очередь вызывает хранимую процедуру sp_table_validation, которая подсчитывает количество строк и, опционально, контрольную сумму опубликованной таблицы. Считается хорошей практикой выполнять ежедневную проверку по ко-

личеству строк и еженедельную по контрольной сумме. Утилита SQL Data Compare идеально подходит для восстановления сбившейся репликации.

Агент распространителя выдает системное сообщение номер 20574, если проверка не удалась или сообщение номер 20575, если проверка прошла успешно. Агент распространителя будет реплицировать изменения на подписчика, даже если проверка показала, что с подписчиком потеряна синхронизация. Полезно настраивать оповещение, реагирующее на системное сообщение номер 20574 и посылающее электронное письмо, сообщение на пейджер или по сети администратору репликации.

К сожалению, такой подход с проверкой может быть использован не всегда. Например, он не будет работать, если в статьях были определены фильтры.

Изменение настроек

Предпочтительно использовать стандартные настройки репликации, за исключением случаев, когда изменение стандартных настроек дает существенный прирост производительности или этого требует архитектура приложения. Однако нельзя утверждать, что изменения будут позитивны для всей топологии в целом без проведения всестороннего тестирования.

Статьи

Статья – это наименьшая единица публикации. Статьей может быть таблица, представление, хранимая процедура или функция. Если статья основана на таблице или представлении, то она может включать все данные или только их часть. Существуют фильтры двух видов. Более распространенными являются статические «WHERE» выражения. В репликации слиянием так же поддерживаются динамические фильтры. Они используются для того чтобы публиковать разные секции данных различным подписчикам. Эти фильтры могут представлять собой простой строчный фильтр или фильтр соединения, где отбор строк для публикации основан на объединении с другими таблицами.

Вместе со статьей можно реплицировать связанные с ней дополнительные объекты, такие как индексы, ограничения целостности, триггеры, расширенные свойства и т.д.

Обновление статей

В репликации слиянием, подписчик может вносить изменения в данные статей. Это создает предпосылки для конфликтов, вызванные тем, что одни и те же данные могут быть изменены разными узлами. Когда агент слияния обрабатывает строки, он использует их историю или «родословную», чтобы идентифицировать конфликты. Если обнаруживается конфликт, то обновление, которое в конечном итоге принимается, выбирается на основе следующих методов:

- Первое изменение, записанное на издателе, побеждает в конфликте.
- Принимается изменение подписчика с наибольшим значением приоритета.
- Пользовательский арбитр конфликтов на основе COM или хранимых процедур.

Агент слияния обнаруживает конфликты, используя столбец lineage системной таблицы MSmerge_contents, который поддерживается автоматически, когда пользователь обновляет строки. Эта колонка содержит информацию о том, какие узлы в топологии репликации слиянием вносили в строку изменения.

Конфликты можно отслеживать на уровне строк или колонок. Чаще используются статьи с отслеживанием изменений на уровне столбцов. Это означает, что конфликтом считается только обновление более чем одним подписчиком одного и того же столбца. Однако иногда бизнес-логика приложения требует рассматривать одновременное изменение любых столбцов в строке как конфликт. В этом случае используется отслеживание изменений на уровне строк.

Программирование репликации

Для развертывания, администрирования и контроля топологии репликации можно использовать Object Explorer в SSMS или Enterprise Manager в предыдущих версиях SQL сервера. Для тех же целей также можно использовать различные программные подходы, такие как TSQL сценарии или RMO (Replication Management Objects) с VB.NET или C#.

Независимо от того, каким способом проводилась первоначальная настройка репликации (например, с использованием мастеров или объектов RMO) полезно иметь полный набор TSQL сценариев развертывания репликации. Эти сценарии можно создать вручную, с помощью мастеров настройки репликации в SQL Server Management Studio или с использованием RMO. Сценарии могут быть сохранены в проекте SQL Server Management Studio. Их ценность заключается в том, что это сохраненная, воспроизводимая последовательность шагов необходимых для полного развертывания топологии репликации. Это позволяет:

- Использовать их для развертывания более чем одного подписчика.
- Упрощает процесс восстановления после серьезного сбоя.
- Обеспечивает некоторый уровень документированности.

При создании сценариев для настройки репликации предпочтительно использовать Windows аутентификацию, чтобы избежать хранения паролей в файлах сценариев. В противном случае нужно как-то обеспечить защищенность этой секретной информации.

Есть и другие типы сценариев, которые относятся к репликации. Они используются в клиентских приложениях. Например, когда по команде пользователя выполняется синхронизация подписок по запросу. Кроме того, сценарии используются для написания специальных бизнес правил, которые выполняются в процессе синхронизации в репликации слиянием.

Дополнительная информация

- Репликация SQL Server
  [*http://msdn2.microsoft.com/ru-ru/library/ms151198.aspx*]
- Все шаги реализации репликации описаны здесь: «Реализация репликации»
  [*http://msdn2.microsoft.com/ru-ru/library/ms151847.aspx*]
- Сведения о настройке и поддержке репликации есть здесь: «Настройка и обслуживание репликации»
  [*http://msdn2.microsoft.com/ru-ru/library/ms151247.aspx*]
- Подробнее об одноранговой репликации можно прочитать здесь: «Одноранговая репликация транзакций»
  [*http://msdn2.microsoft.com/ru-ru/library/ms151196.aspx*]
- Некоторые из широко распространенных проблем и опасностей репликации рассматривается в статье «Data Replication as an Enterprise SOA Antipattern» в интереснейшем журнале «Microsoft Architecture Journal»
  [*http://msdn2.microsoft.com/ru-ru/library/bb245678.aspx*]
  [*http://msdn2.microsoft.com/ru-ru/library/bb410935.aspx*]

SQL Server 2005: Графический интерфейс репликации

По материалам статьи Пола Ибизона (Paul Ibison)
«SQL Server 2005: The Replication GUI»
Перевод Яна Дмитриевича Либермана

Эта статья описывает изменения в репликации с точки зрения графического интерфейса (GUI) в SQL Server 2005. Если вы еще не успели познакомиться с новой версией SQL Server, то вам, вероятно, будет интересно узнать что, вместо «Enterprise Manager» теперь используется «Microsoft SQL Server Management Studio». В отличие от многих других задач, для управления репликацией администраторы интенсивно используют графический интерфейс. Учитывая это, администраторам репликации будет интересно понять, как выполнять свои повседневные задачи, используя новый графический интерфейс. Я планирую написать цикл статей посвященных новым возможностям репликации в SQL Server 2005, однако в этой статье основное внимание уделяется тому, как использовать новый графический интерфейс для решения базовых задач репликации.

Microsoft SQL Server Management Studio

Если мы посмотрим на «Object Explorer» в Management Studio, то сразу увидим некоторые изменения. На рис. 1 представлена публикация транзакций «TransParentsTable» в базе данных «PaulsPublisher» и подписка на эту публикацию в базе данных «PaulsSubscriber». Итак, что изменилось? Во-первых, нет руки в значке базы данных, указывающей, что для базы данных разрешено применение репликации. Во-вторых, в Enterprise Manager, публикация была доступна в трех разных местах: в папке внутри базы данных, в папке «Replication» и в мониторе репликации («Replication Monitor»). На рис. 1 видно, что теперь публикация доступна только в папке «Replication». Безусловно, это большое упрощение, которое в частности устраняет раздражающее несоответствие между контекстными меню одной публикации в трех разных местах в Enterprise Manager.

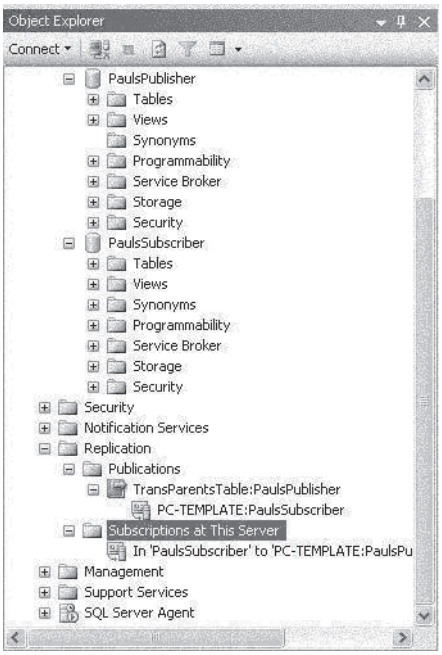


Рис. 1.

Выбор публикации (например, нажатием на ней левой кнопки мыши) делает доступной в правой части окна область «Summary» (рис. 2). Здесь отображается информация о текущем состоянии и последнем событии агентов моментального снимка и чтения журнала.

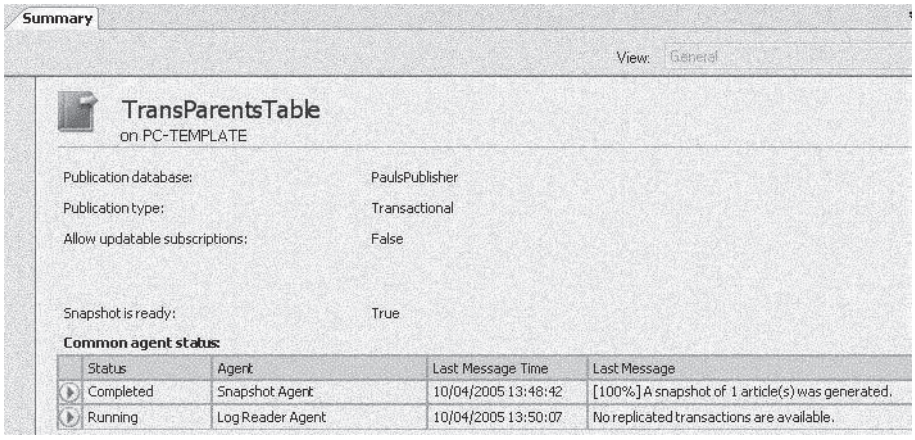


Рис. 2.

Информация по агенту распространителя в настоящее время недоступна, но предположительно она будет добавлена. Заметим, что содер-

жимое области «Summary» – это на самом деле HTML-страница. Поэтому управлять агентами с помощью контекстного меню нельзя. То есть это всего лишь отчет. Это отличается от того, что было в Enterprise Manager, так как непосредственно агенты больше не доступны через папки replication/publication. Разумеется, мы понимаем, что по существу агенты это задания (job) запускающие соответствующие выполняемые файлы и, следовательно, они доступны через папку Jobs в SQL Server Agent. Однако это связано с поиском нужных заданий среди множества других. Поэтому, для доступа непосредственно к агентам, мы должны использовать монитор репликации (replication monitor), который существенно изменился, и сейчас является самостоятельным приложением (см. ниже).

Если щелкнуть правой кнопкой мыши по папке «Replication», то будет отображено следующее контекстное меню:

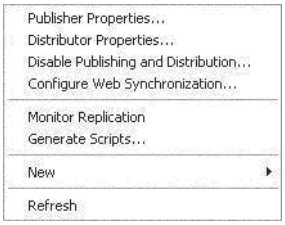


Рис. 3.

Это меню похоже на контекстное меню папки Replication Monitor в Enterprise Manager, а также меню доступное через Tools/Replication. Последнее теперь недоступно – так что сейчас приведенное выше меню это единственный способ получить доступ к свойствам издателя и распространителя. В целях упрощения управление свойствами издателя и распространителя разделено на два разных пункта. Контекстное меню публикации дает доступ к ее свойствам, так же как и в SQL Server 2000. Хотя стиль окон отличается от SQL Server 2000 (окна свойств с большим количеством закладок), сами опции в основном те же самые.

Монитор репликации

Чтобы запустить монитор репликации, нужно выбрать соответствующий пункт контекстного меню папки «Replication» или папки публикации. Сейчас это автономное средство и поэтому первое, что нужно сделать – это зарегистрировать издатель (на приведенном ниже рисунке это PC-Template).

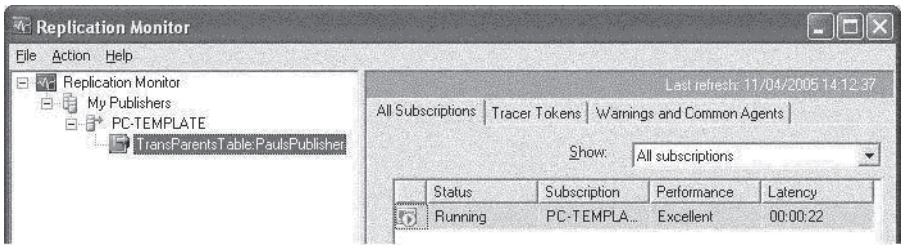
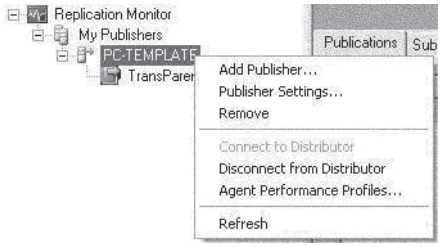


Рис. 4.

Изменение частоты обновления для монитора репликации доступно через контекстное меню издателя. Теперь поддерживаются разные настройки для разных издателей, а не одна настройка на всех, как было ранее. С помощью того же меню можно создавать и редактировать профили агентов:



Закладки справа предоставляют доступ к агентам репликации. Закладка «All Subscriptions» относится к агентам распространителя и слияния, а закладка «Warnings and Common Agents» предоставляет доступ к агенту чтения журнала и агенту моментального снимка. Непосредственно на закладке отображается текущее состояние, в то время как всю историю можно посмотреть, если выбрать пункт «View Details» в контекстном меню соответствующего агента. С помощью этого же контекстного меню (пункт «Performance Profile») также можно получить доступ к профилям агента.

Рис. 6.

Другие настройки

Похоже, что некоторые агенты, например агент очистки распространителя, доступен только через папку Jobs в SQL Server Agent. Это вызывает сожаление, однако с другой стороны это, безусловно, не то, что часто требуется настраивать вручную. Поддержка трансформируемых подписок была удалена из графического интерфейса, так как эта функциональность в SQL Server 2005 объявлена устаревшей (для ее использования необходимо установить службы DTS SQL Server 2000). Также как и в Enterprise Manager, реализованы генерация сценариев и проверка подписок. По-видимому, функции «Explore the latest snapshot folder» (которая была в Enterprise Manager) больше нет и поэтому нужно напрямую использовать проводник Windows.

Заключение

Надеюсь, что эта статья поможет администраторам репликации разобраться с новым графическим интерфейсом репликации, а также поможет понять как привычные задачи решаются в SQL Server 2005. В статье обойдены вниманием некоторые новые функциональные возможности (в том числе относящиеся к графическому интерфейсу). О них я планирую рассказать в будущих статьях.

Вы все еще используете предварительные и заключительные сценарии моментального снимка?

По материалам статьи Пола Ибизона (Paul Ibison)
«Are you using unnecessary pre/post-snapshot scripts?»
Перевод Яна Дмитриевича Либермана

Введение

В SQL Server 2000 мы часто использовали предварительные и заключительные сценарии моментального снимка. Предварительные сценарии выполняются на подписчике перед применением моментального снимка, а заключительные сценарии, соответственно, после. Эти сценарии используются для различных целей. Например, с помощью предварительных сценариев можно создать пользователя, который будет являться владельцем реплицируемых таблиц или создать файловые группы, в которые эти таблицы должны быть помещены. Часто использование предварительных и заключительных сценариев было связано с поддержкой ограничений ссылочной целостности. Новые возможности, добавленные в репликацию транзакций и репликацию моментальных снимков в SQL Server 2005, во многих случаях позволяют сделать то же самое без использования дополнительных сценариев.

Если реплицируемые статьи принадлежат к одной публикации и в параметрах создания схемы для статьи включена репликация ограничений внешних ключей (параметр @schema_option процедуры sp_addarticle содержит 0x200 или 0x20000), то дополнительные сценарии не нужны. Связи между таблицами создаются автоматически, а таблицы при необходимости удаляются в правильном порядке. Однако, что если в репликации участвует более одной публикации? В этом случае удаление таблиц на подписчике может привести к ошибке связанной с наличием внешних ключей на статьи (таблицы) входящие в другие публикации. Также на подписчике могут существовать таблицы, которые не входят ни в одну публикацию и связанные с реплицируемыми таблицами. Здесь мы столкнемся с той же проблемой. Она обычно решалась следующим образом – в предварительном сценарии внешние ключи удаля-

лись, а в заключительном сценарии создавались заново. Создавать эти сценарии приходилось вручную, хотя это и несложно.

Изменения в моментальном снимке в SQL Server 2005

В SQL Server 2005 появились новые файлы сценариев с расширением «.pre». Их можно увидеть в папке моментальных снимков, конечно, если посмотреть туда до того, как они будут удалены. Вы могли видеть эти файлы и раньше, если использовали статьи, созданные с параметром @pre_creation_cmd (в sp_addarticle) равным «truncate». Так что они не являются чем-то абсолютно новым. Однако сейчас файлы «.pre» создаются отдельно на каждую статью, и их содержимое тоже изменилось. Туда помещается вызов системной хранимой процедуры sys.sp_MSdropfkreferencingarticle, которая сохраняет информацию о созданных на подписчике ограничениях ссылочной целостности в следующих таблицах:

```
dbo.MSsavedforeignkeys
dbo.MSsavedforeignkeycolumns
dbo.MSsavedforeignkeyextendedproperties
```

Как только нужная информация успешно сохранена, внешние ключи удаляются. После того, как моментальный снимок применен на подписчике, удаленные внешние ключи восстанавливаются по информации из вышеупомянутых таблиц. За это отвечает системная хранимая процедура sp_MSrestoresavedforeignkeys, которая выполняется непосредственно агентом распространителя (никаких дополнительных сценариев, соответствующих «.pre» сценариям для этого не создается).

Заключение

Взгляните на предварительные и заключительные сценарии моментального снимка, которые вы используете. Если в них содержится код, предназначенный для поддержки ограничений ссылочной целостности, то очень вероятно, что они делают работу, которую SQL сервер и так делает по умолчанию. В этом случае можно просто от них отказаться. Интересно, что описанное в этой статье поведение SQL сервера нельзя изменить. Хотя, честно говоря, сложно представить ситуацию, в которой может потребоваться менять это поведение. Но, если по каким-то причинам вас такое поведение SQL сервера не устраивает, вероятно, небольшого заключительного сценария будет достаточно, чтобы решить проблему.

BCP-секционирование в SQL Server 2005

По материалам статьи Пола Ибизона (Paul Ibison)
«SQL Server 2005 BCP Partitioning»
Перевод Яна Дмитриевича Либермана

Введение

В SQL Server 2000 в процессе генерации моментального снимка, когда статьи экспортируются в файловую систему с помощью BCP (в папку моментальных снимков), всегда создается по одному файлу с данными для каждой статьи.

В SQL Server 2005, если вы посмотрите на результат генерации моментального снимка, то вероятно будете удивлены, увидев в папке моментальных снимков по нескольку файлов для каждой статьи, где в каждом файле будет содержаться только часть данных.

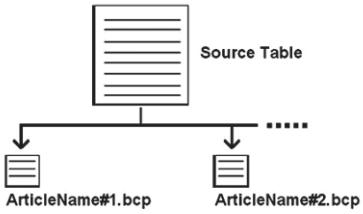


Рис. 1.

Очевидно, что в процесс формирования моментального снимка были внесены существенные изменения. Причем в документации (BOL) это никак не отражено. Я буду называть этот процесс «BCP секционирование» (термин взят мной из обсуждения на Microsoft Replication Newsgroup, форуме, посвященном репликации). Эта статья рассказывает о том, для чего нужно BCP секционирование, что следует от него ждать и что делать, если что-то будет работать не так.

Зачем это нужно?

Есть несколько преимуществ, которые дает BCP секционирование. Во-первых, в процессе применения моментального снимка на подписчике, может произойти сбой (например, из-за проблем в сети). Для SQL 2000 это означает, что моментальный снимок должен быть применен заново в

полном объеме. Кроме того, в случае параллельного применения моментальных снимков («concurrent snapshot») все будет выполнено в рамках одной транзакции. Но если и на распространителе и на подписчиках установлен SQL Server 2005, то процесс разбивается на несколько меньших по объему операций. А каждая секция применяется в рамках отдельной транзакции. Это означает, что если произойдет сбой, то не надо будет делать всю работу заново, а можно будет продолжить с того места, где возникли проблемы. Для больших таблиц это может означать гигантское ускорение.

Еще одно преимущество заключается в том что «разрастание» журнала транзакций будет не таким большим (если предположить что у нас используется простая модель восстановления или если процесс пересекается по времени с резервным копированием журнала транзакций, например, если он запускается по расписанию).

Для многопроцессорных машин, возникает дополнительная возможность распараллеливания процесса BCP. Конечно, распараллеливание было и в SQL Server 2000, но только на уровне статей (при условии, что их несколько), но не для отдельных таблиц.

Очевидно, что все эти преимущества также применимы и к случаю создания исходного моментального снимка.

Тестирование

Для того, что бы посмотреть, сколько файлов будет создано, и понять, как работает алгоритм, я создал таблицу TestBCP и заполнил ее данными (код приведен далее).

```
CREATE TABLE TestBCP(id int not null,
fullname varchar(100) COLLATE SQL_Latin1_General_CP1_CI_AS null,
CONSTRAINT PK_TestBCP PRIMARY KEY CLUSTERED (id ASC) ON PRIMARY
) ON PRIMARY

declare @id int
set @id = 1
while @id < 1501
begin
INSERT INTO testbcp(id, fullname)
values(@id, 'Name ' + cast(@id as varchar(100)))
set @id = @id + 1
end
```

В ходе экспериментов на своей машине, первоначально, я нашел простую зависимость количества результирующих файлов от числа строк в таблице, но дальнейшие исследования давали совершенно необъяснимый результат. То есть при изменении размера таблицы количество файлов менялось, каким то непонятным образом. Почему так происходило, полностью описано в разделе «диагностика».

Результаты

(a) Для 8 и менее процессоров, формула для расчета числа секций (файлов) будет иметь следующий вид:

КОЛИЧЕСТВО_СЕКЦИЙ = КОЛИЧЕСТВО_ПРОЦЕССОРОВ x 4

Так как на моей машине установлено 4 процессора, то я ожидал, что будет создано 16 секций. Так оно в точности и получилось. Надо сказать, что даже для 1 процессора создается 4 секции. Это подтверждает тот факт, что многопоточность лишь одна из причин реализации этого алгоритма. Стоит отметить, что параметр –BcpBatchSize агентов моментального снимка и распространителя просто указывает, как часто следует выдавать сообщение о ходе работ и абсолютно не влияет на число секций.

(b) Установлен порог в 1000 строк, после которого, включается алгоритм секционирования.

(c) Данные равномерно распределяются между секциями (файлами данных)

Диагностика

(a) Почему создается «неправильное» количество секций?

Как я уже упоминал, во многих случаях, то, что я видел на практике, не соответствовало сформулированным выше правилам. В некоторых случаях я ожидал увидеть много секций, но все данные помещались в один файл. В других случаях, наоборот, в таблице было меньше 1000 строк, но создавалось много файлов притом, что большинство из них были пустыми.

Запуск DBCC SHOWSTATS позволил понять проблему. Ниже приведен результат выполнения команды DBCC SHOW_STATISTICS (“testbcp”, “PK_TestBCP”). Стало понятно, что статистики были неактуальными, а процессы репликации, самостоятельно их не обновляют.

	Name	Update	Rows	Rows Sampled	Steps	Desity	Average key length	String Index
1	PK_TestBCP	Oct 23 2008 2:37PM	998	998	200	1	4	NO

Рис. 2.

Значение в колонке «Rows» приведенной выше таблицы, должно в точности совпадать с тем, что возвращает команда «select count(*) from testbcp». Во всех случаях запуск команды «UPDATE STATISTICS testbcp» делало значение корректным и, следовательно, количество создаваемых секций соответствовало правилам.

(b) Отключение BCP секционирования

Что бы отключить BCP секционирование, нужно, как показано на рисунке ниже, добавить недокументированный параметр «-EnableArticleBcpPartitioning 0» для агента моментального снимка. Как результат, только один файл будет создан (также как это было в SQL Server 2000).

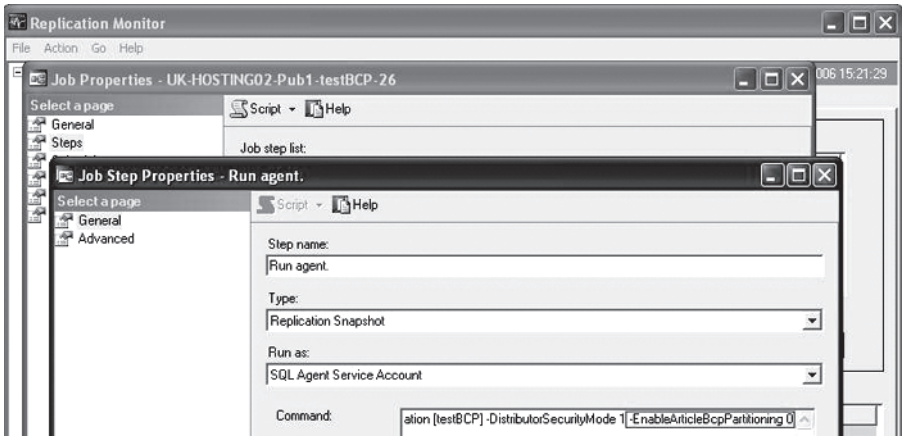


Рис. 3.

Зачем вам может понадобиться отключать столь полезную функцию? Например, если используется параллельная обработка моментальных снимков (включено по умолчанию для SQL2005), а процессор, диск или пропускная способность сети могут стать узким местом, в случае если попытаться еще больше распараллелить процесс при помощи BCP секционирования.

(c) Обеспечение неполного протоколирования

Для таблиц, которые приводят к сильному росту журнала транзакций, некоторые администраторы, для того чтобы этот рост ограничить, любят включать модель восстановления с неполным протоколированием. Однако когда мы имеем дело с несколькими секциями, это не всегда дает желаемый результат. Для того, что бы уменьшить рост журнала транзакций вы должны использовать параметр агента распределителя -MaxBcpThreads > 1 и при этом убедиться, что результирующая таблица не имеет индексов, перед тем как агент распределитель доставит моментальный снимок. Или использовать -MaxBcpThreads = 1 для того, что бы отключить распараллеливание, хотя безусловно, это может снизить производительность.

Заключение

BCP секционирование в SQL Server 2005 – это новая функциональность, которая дает нам ряд существенных преимуществ. Благодаря ей, например, процесс инициализации, несомненно, стал более гибким и как правило быстрым. Мы можем даже не знать, что в SQL Server 2005 алгоритм поменялся и просто принять то, что каким то образом система стала работать лучше и быстрей. Но я надеюсь, что эта статья пролила немного больше света на то, что происходит «за сценой», и поможет вам в случае, если вдруг возникнут какие то проблемы.

Посредники, учетные данные и подсистемы репликации

По материалам статьи Пола Ибисона (Paul Ibison)
«Replication Proxies, Credentials and Subsystems»
Перевод Ильдара Ахметовича Даутова

Вступление

В этой статье я хотел бы рассмотреть новую опцию безопасности заданий репликации. Надеюсь это даст лучшее понимание того, как это действительно работает и поможет в решении возникших проблем. Что за новая опция, о которой я говорю? Если вы создавали публикацию в SSMS с помощью мастера, то наверняка заметили экран, аналога которого нет в SQL Server 2000. На этом экране есть опция «Run under the following Windows account»:

Рис. 1

Показанный экран связан с агентом репликации моментальным снимком (snapshot) и следовательно с заданием репликации моментальным снимком. Аналогичные экраны есть в свойствах существующей публикации и при создании подписчика.

Другими словами, эта новая опция применима ко всем агентам, участвующим в репликации. В хранимых процедурах, создающих задания репликации, также появились два новых параметра «@job_login» и «@job_password», например, в sp_addpublication_snapshot.

Для чего использовать новую опцию? Если вернуться к SQL Server 2000, то как там обеспечивалась безопасность заданий репликации? Задания выполнялись от имени учетной записи, которую использовал агент SQL Server и, например, учетная запись агента SQL Server должна была иметь права на каталог с моментальным снимком у издателя при pull-подписке и, соответственно, быть доменной учетной записью (мы не принимаем в расчет те редкие установки репликации, которые работают на одной машине).

В SQL Server 2005 теперь есть возможность использовать учетную запись, отличную от учетной записи агента SQL Server. Почему это полезно? В SQL Server 2000 агент SQL Server часто выполняется под высокопривилегированной доменной учетной записью. Иногда эта учетная запись –администратор домена, иногда – одна и та же учетная запись используется и на издателе, и на подписчике и является локальным администратором на этих серверах.

Таким образом, пользователи, получающие доступ на выполнение сценариев от имени такой высокопривилегированной учетной записи, могут нанести вред системе. Один из возможных путей сделать это состоит в использовании хранимой процедуры sp_addscriptexec, которая используется в репликации для передачи и выполнения скрипта на подписчике. Эта хранимая процедура полезна, когда необходимо сделать изменения без реинициализации, например, создать новый индекс или изменить хранимую процедуру.

В любом случае, чтобы выполнить sp_addscriptexec необходимо быть членом роли db_owner и только. Поэтому, если репликация использует Windows-пользователя только с требуемыми правами, sp_addscriptexec не сможет нанести вред системе. Таким образом реализуется принцип минимальных привилегий.

Как это работает: (а) посредники и учетные данные

Чтобы понять суть происходящего, нам необходимо познакомится с новыми понятиями «учетные данные» (credentials) и «посредники» (proxies).

Из BOL: «**Учетные данные** – это запись, которая содержит информацию аутентификации, необходимую для доступа к ресурсу, внешнему

по отношению к SQL Server. Большинство учетных данных включают пользователя Windows и пароль. **Посредник** агента SQL Server управляет безопасностью шагов заданий, отличных от TSQL. Каждый посредник соответствует учетным данным».

Итак, мы ожидаем, что пользователь «groupbases\ibisonp» на предыдущем экране станет учетными данными, и поэтому где-то должен существовать посредник, который свяжет его с заданием репликации моментальным снимком. К счастью, оба эти объекта представлены графически в SSMS, и я раскрыл соответствующие узлы на рисунке, показанном ниже:

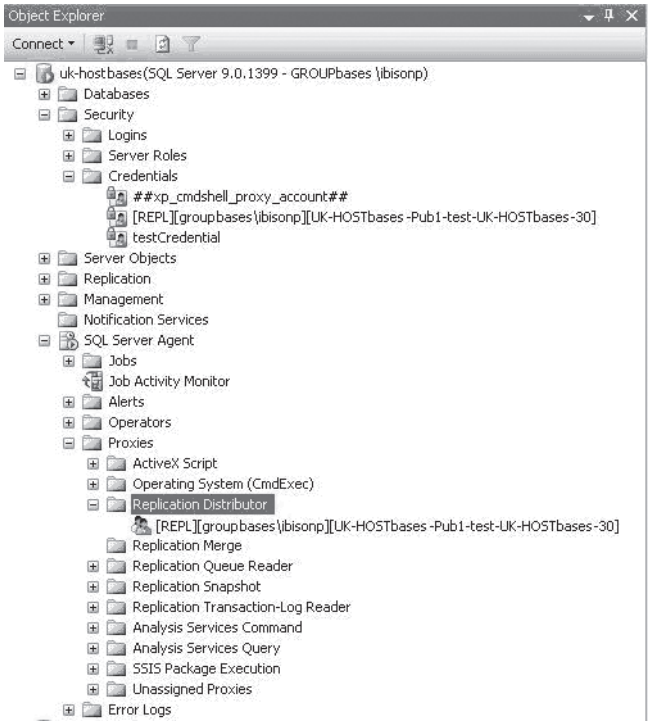


Рис. 2

Длинное имя («[REPL][groupbases\ibisonp][UK-HOSTbases-Pub1-test-UK-HOSTbases-30]»), использованное в учетных данных и посреднике действительно полезно – оно показывает связь с репликацией («[REPL]»), олицетворенным пользователем Windows («[groupbases\ibisonp]») и фактическим заданием («[UK-HOSTbases-Pub1-test-UK-HOSTbases-30]»).

В примере, приведенном выше, я выбрал учетную запись «groupbases\ibisonp». Если это нас не устраивает, мы можем изменить это решение для задания репликации, выбрав новое значение в выпадающем списке «Run as:», например, «SQL Server Service Account», что вернет нас к поведению SQL Server 2000.

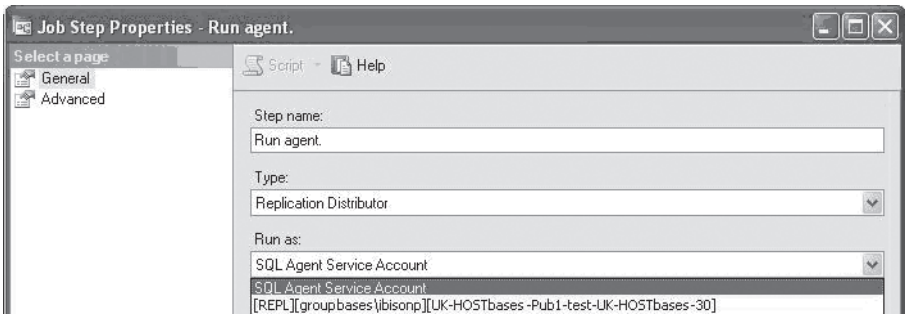


Рис. 3

Можно создать новые учетные данные и посредники вручную с помощью TSQL («create credential», «sp_add_proxy» и «sp_grant_proxy_to_subsystem»). После создания они будут появляться в выпадающем списке «Run as:». Другой способ создать и сопоставить новый посредник – использовать хранимую процедуру sp_changepublication_snapshot.

Суммируя вышесказанное, мы можем создать посредников и учетные данные:

(a) с помощью опции мастера «Run under the following windows account»

(b) с помощью Create Credential, sp_add_proxy and sp_grant_proxy_to_subsystem

(c) с помощью sp_changepublication_snapshot.

Как это работает: (b) подсистемы репликации

Теперь мы знаем о посредниках и учетных данных, и можем взглянуть на то, как они используются различными процессами. Агент репликации моментальным снимком и подсистема репликации – два основных компонента. Агент репликации моментальным снимком (snapshot.exe) отвечает за генерацию файлов снимка, а подсистема репликации (sqlrepss90.dll) отвечает за запуск агента (чтобы увидеть больше информации о каждой подсистеме, соответствующих dll и программах, можно выполнить запрос «select * from msdb.dbo.syssubsystems»). Следующий список показывает эту взаимосвязь.

SQL Server Agent → Задание → Подсистема репликации → Посредник → Учетные данные → Snapshot agent

Выводы

Надеюсь, эта статья поможет понять новые опции безопасности, которые появились в мастерах репликации. Стоит придерживаться принципа минимальных привилегий, в противном случае злонамеренный пользователь репликации сможет получить дополнительные возможности, и в большинстве случаев делать все, что он захочет на компьютере подписчика. Настройка репликации с использованием посредников и учетных данных проста (и автоматически, и вручную), и выполнив ее однажды, вы поймете простоту и красоту решения.

Использование утилиты TableDiff в репликации

Автор Григорий Петрович Корнилов

Введение

В этой статье подробно описывается использование новой утилиты TableDiff для решения проблем связанных с несогласованностью данных в двух таблицах, находящихся в участвующих в репликации базах данных. Обычно это связано с репликацией, когда данные подписчика и издателя не совпадают, однако, эта утилита может использоваться в любой похожей ситуации. Утилиту TableDiff можно запускать как автономную, просто скопировав исполняемый файл TableDiff.exe на другую машину – единственное требование состоит в том, что для корректной работы утилита нуждается в установке правильной версии .NET Framework. Возможно, например, использовать эту утилиту для того, чтобы сравнивать базы данных SQL Server 2000.

Эта утилита используется для сравнения данных в двух таблицах или представлениях, вот, что о ее возможностях написано в электронной документации по SQL Server.

- Построчное сравнение исходной таблицы в экземпляре Microsoft SQL Server, выступающем в качестве издателя репликации, и целевой таблицы в одном или нескольких экземплярах SQL Server, выступающих в качестве подписчиков репликации.
- Быстрое сравнение, сравнивающее только схемы (структуры) и количество строк.
- Сравнение целевых таблиц одновременно на нескольких целевых серверах.
- Сравнение на уровне столбцов.
- Формирование сценариев Transact-SQL для исправления несоответствий на целевом сервере и сведения исходной и целевой таблиц.

Запись результатов операции в консоль, файл вывода или в таблицу целевой базы данных.

Прежде всего, где расположена данная утилита? Эта утилита (исполняемый файл -tablediff.exe) – программа, запускаемая из командной строки, которая обычно располагается в папке C:\Program Files\Microsoft SQL Server\90\COM.

Синтаксис

```
tablediff
[ -? ] |
{
        -sourceserver source_server_name[\instance_name]
        -sourcedatabase source_database
        -sourcetable source_table_name
    [ -sourceschema source_schema_name ]
    [ -sourcepassword source_password ]
    [ -sourceuser source_login ]
    [ -sourcelocked ]
        -destinationserver destination_server_name[\instance_name]
        -destinationdatabase subscription_database
        -destinationtable destination_table
    [ -destinationschema destination_schema_name ]
    [ -destinationpassword destination_password ]
    [ -destinationuser destination_login ]
    [ -destinationlocked ]
    [ -b large_object_bytes ]
    [ -bf number_of_statements ]
    [ -c ]
    [ -dt ]
    [ -et table_name ]
    [ -f [ file_name ] ]
    [ -o output_file_name ]
    [ -q ]
    [ -rc number_of_retries ]
    [ -ri retry_interval ]
    [ -strict ]
    [ -t connection_timeouts ]
}
```

Аргументы

[-?]

Возвращает список поддерживаемых параметров.

-sourceserver *source_server_name*[*instance_name*]

Имя исходного сервера. Для обращения к экземпляру сервера SQL Server по умолчанию укажите *source_server_name*. Для обращения к именованному экземпляру SQL Server укажите *source_server_name**instance_name*.

-sourcedatabase *source_database*

Имя базы данных-источника.

-sourcetable *source_table_name*

Имя проверяемой исходной таблицы или представления.

-sourceschema *source_schema_name*

Владелец схемы исходной таблицы. Владельцем таблицы по умолчанию считается **dbo**.

-sourcepassword *source_password*

Пароль для имени входа, используемого для подключения к исходном серверу с помощью проверки подлинности SQL Server.

Примечание о безопасности: по возможности указывайте учетные данные во время выполнения. Если необходимо хранить учетные данные в файле сценария, необходимо обеспечить его безопасность, чтобы предотвратить несанкционированный доступ.

-sourceuser *source_login*

Имя входа, используемое для подключения к исходному серверу с помощью проверки подлинности SQL Server. Если имя *source_login* не указано, для соединения с исходным сервером используется проверка подлинности Windows.

-sourcelocked

Исходная таблица блокируется в ходе сравнения при помощи табличных подсказок TABLOCK и HOLDLOCK.

-destinationserver *destination_server_name*[*instance_name*]

Имя целевого сервера. Для обращения к экземпляру сервера SQL Server по умолчанию укажите *destination_server_name*. Для обращения к именованному экземпляру SQL Server укажите *destination_server_name**instance_name*.

-destinationdatabase *subscription_database*

Имя целевой базы данных.

-destinationtable *destination_table*

Имя целевой таблицы или представления.

-destinationschema *destination_schema_name*

Владелец схемы целевой таблицы. Владельцем таблицы по умолчанию считается **dbo**.

-destinationpassword *destination_password*

Пароль для имени входа, используемого для подключения к целевому серверу с помощью проверки подлинности SQL Server.

Примечание о безопасности: по возможности указывайте учетные данные во время выполнения. Если необходимо хранить учетные данные в файле сценария, необходимо обеспечить его безопасность, чтобы предотвратить несанкционированный доступ.

-destinationuser *destination_login*

Имя входа, используемое для подключения к целевому серверу с помощью проверки подлинности SQL Server. Если имя *destination_login* не указано, для соединения с исходным сервером используется проверка подлинности Windows.

-destinationlocked

Целевая таблица блокируется в ходе сравнения при помощи табличных подсказок TABLOCK и HOLDLOCK.

-b *large_object_bytes*

Число байтов для сравнения столбцов, содержащих данные типа больших объектов, к которым относятся: **text**,**ntext**, **varchar(max)**, **nvarchar(max)** и **varbinary(max)**. Значение по умолчанию – 1 000 байт, максимальное значение – 8 000 байт. Любые данные, размер которых превышает значение *large_object_bytes*, игнорируются при сравнении.

-c

Сравнение на уровне столбцов.

-dt

Удаление таблицы результатов, указанной в аргументе *table_name*, если она уже существует.

-et *table_name*

Имя создаваемой таблицы результатов. Если таблица уже существует, необходимо использовать аргумент **-DT**, иначе возникнет ошибка.

-f –bf #num

Формирует сценарий Transact-SQL, по которому обеспечивается конвергенция таблицы на целевом сервере и таблицы на исходном сервере. Если возможность задать максимальное количество операций в создаваемых файлах (например, если **–bf 3** и будет будут найдены расхождения по 10-ти записям, то будет создано 10/3=4 файла).

-o *output_file_name*

Полное имя и путь файла вывода. Если не указать имя файла, то по умолчанию сохранится в файл с именем, подобным показанному ниже:

C:\Program Files\Microsoft SQLServer\90\COM\
DIFFIX.632722665324687500

-q

Быстрое сравнение, сравнивающее только схемы и количество строк.

-t *connection_timeouts*

Период времени ожидания в секундах для соединения с исходным сервером и каждым целевым сервером.

Замечания

Программа **tablediff** не может использоваться для обращения к серверам, отличным от SQL Server.

Таблицы в нескольких подписках могут сравниваться одновременно путем отделения каждого набора параметров подписки запятой (,) или знаком, указанным в аргументе *custom_destination_delimiter*. Для каждой подписки необходимо указывать одинаковый набор параметров подписки

Разрешения

Для сравнения таблиц на сравниваемые объекты таблиц необходимо иметь разрешения SELECT ALL.

Для использования параметра **-et** необходимо быть членом фиксированной роли базы данных **db_owner** или как минимум иметь разрешение CREATE TABLE в базе данных подписки и разрешение ALTER на схему конечного владельца на целевом сервере.

Для использования параметра **-dt** необходимо быть членом фиксированной роли базы данных **db_owner** или как минимум иметь разрешение ALTER на схему конечного владельца на целевом сервере.

Для использования параметра **-o** необходимо иметь разрешения на запись в указанном каталоге.

Дополнение

Теперь, когда вы прочитали основу (которая есть в статье по данной утилите из BOL на *www.microsoft.ru/rus*), я добавлю немного от себя:

Утилита для идентификации записей требует наличия на таблицах первичных ключей, или полей со свойством identity, rowguid или уникальных ключей, соответственно при их отсутствии вы получите сообщение об ошибке:

Действия утилиты в MS SQL Server можно просмотреть, используя стандартный инструмент SQL Server Profiler.

Опции -destinationlocked и -sourcelocked

Почему есть необходимость в блокировке во время выполнения работы утилиты, и как работает TABLEDIFF по умолчанию? При нахождении записей, которые отличаются, по умолчанию утилита использует примерно следующий запрос:

```
SELECT
[dbo].[vwsourcedata].[Id],BINARY_CHECKSUM([dbo].[vwsourcedata].[Data],[dbo].[vwsourcedata].[Id])
as MShash_54267293 FROM
[dbo].[vwsourcedata] WITH (READUNCOMMITTED) ORDER BY [dbo].[vwsourcedata].[Id]
```

Здесь мы видим, что производилось чтение с уровнем изоляции READUNCOMMITTED, который не всегда является достаточным. При использовании опций утилита произведет дополнительный, похожий на приведенный в следующем примере запрос, который приведет к блокировке данных на более высоком уровне:

```
SELECT TOP 1 FROM  [dbo].[vwdestantiondata] WITH (TABLOCK, HOLDLOCK)
```

Использование представлений

Использование представлений, а не таблиц позволяет нам производить операции, когда исходные и целевые данные хранятся в различных структурах. В самом простом случае столбцы таблиц имеют разные наименования, однако возможны более сложные варианты:

- Целевые данные могут вычисляться по сложной функции от исходных данных (целевой строковый столбец является конкатенацией нескольких исходных строковых столбцов).
- Необходимость применения утилиты только на исходных данных с определенными критериями (фильтрация исходных данных по условию).
- Применение утилиты к объекту, который как в исходных данных, так и в целевых данных хранится в нескольких таблицах или даже в нескольких базах данных (и в данном случае решающую роль играет применение триггеров типа INSTEAD OF)

Варианты результатов

Создание объектов базы данных и заполнение их данными:

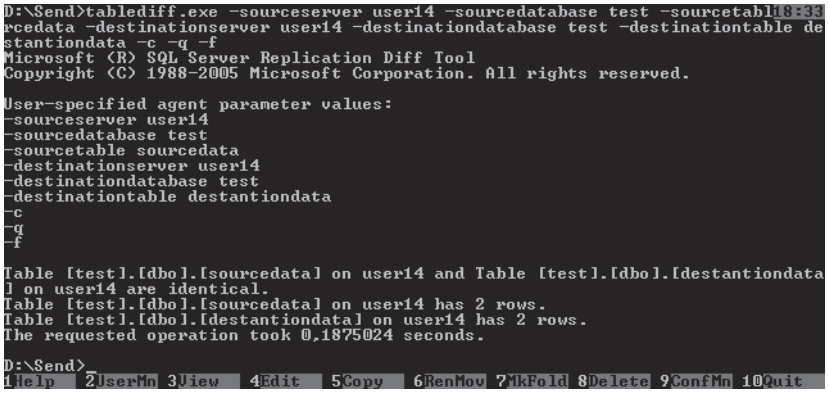
```
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO

CREATE TABLE [dbo].[SourceData](
    [Id] [int] NOT NULL,
    [Data] [varchar](50) COLLATE Cyrillic_General_CI_AS NOT NULL
 CONSTRAINT [PK_SourceData] PRIMARY KEY CLUSTERED
(
    [Id] ASC
)WITH (IGNORE_DUP_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
GO

CREATE TABLE [dbo].[DestantionData](
    [Id] [int] NOT NULL,
    [Data] [varchar](100) COLLATE Cyrillic_General_CI_AS NOT NULL
```

```
 CONSTRAINT [PK_DestantionData] PRIMARY KEY CLUSTERED
(
    [Id] ASC
)WITH (IGNORE_DUP_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
GO

DELETE FROM [SourceData]
DELETE FROM [DestantionData]

INSERT INTO [SourceData] (Id, Data) values (1, “Данные1”)
INSERT INTO [SourceData] (Id, Data) values (2, “ Данные 2”)
INSERT INTO [DestantionData] (Id, Data) values (2, “Данные1”)
INSERT INTO [DestantionData] (Id, Data) values (3, “Данные2”)
```

Запуская:

```
tablediff.exe -sourceserver user14 -sourcedatabase test -sourcetable sourcedata -
destinationserver user14 -destinationdatabase test -destinationtable destantiondata -c -q
–f
```

Получаем:

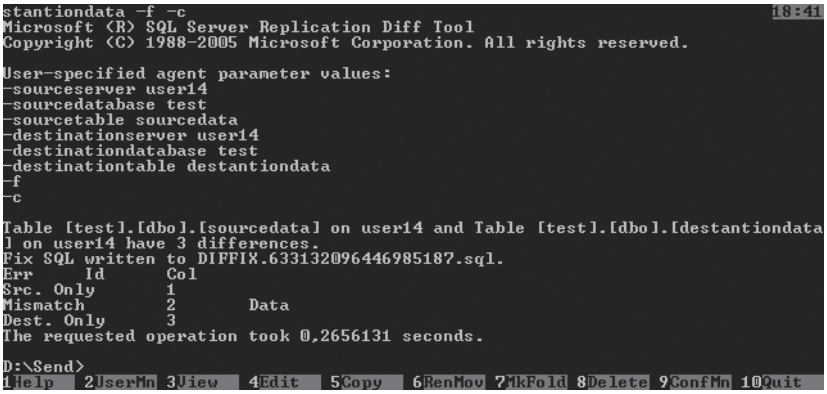


Во-первых мы видим параметры запуска (они важны, так как запишутся в файл при использовании опции **-o** и мы о них будем помнить), далее, утверждение об идентичности надо понимать в контексте выполнения (в данном случае используется опция **-q**, которая имеет приоритет над **-c**, хотя задана она была позже) – таблицы имеют одинаковое количество записей. В конце нам показано время выполнения.

Запуская:

```
tablediff.exe -sourceserver user14 -sourcedatabase test -sourcetable sourcedata -
destinationserver user14 -destinationdatabase test -destinationtable destantiondata –f -c
```

Получаем:



В контексте выполнения (в данном случае используются опции **-c -f**) – таблицы имеют 3 расхождения; SQL-операции для их устранения были записаны в файл; также нам представлены общие данные:

- Тип изменения (Src. Only – есть запись в источнике, но нет в целевом объекте; Mismatch – записи отличаются, и также указывается столбец с несовпадающими данными; Dest. Only – есть запись в целевом объекте, но нет в источнике)
- Номер операции.
- Столбец с расхождением данных.

В файле DIFFIX...sql видим:

```
-- Host: user14
-- Database: [test]
-- Table: [dbo].[destantiondata]
INSERT INTO [dbo].[destantiondata] ([Data],[Id]) VALUES ('Данные1',1)
UPDATE [dbo].[destantiondata] SET [Data]='Данные2' WHERE [Id] = 2
DELETE FROM [dbo].[destantiondata] WHERE [Id] = 3
```

Далее изменим структуру и данные:

```
ALTER TABLE dbo.SourceData ADD Data2 varchar(50) COLLATE Cyrillic_General_CI_AS
NULL

DELETE FROM [SourceData]
DELETE FROM [DestantionData]

INSERT INTO [SourceData] (Id, Data, Data1) values (1, 'Данные1')
INSERT INTO [SourceData] (Id, Data, Data1) values (2, 'Данные', '2')
INSERT INTO [DestantionData] (Id, Data) values (2, ' Данные1')
INSERT INTO [DestantionData] (Id, Data) values (3, ' Данные2')


SET ANSI_NULLS ON
```

```
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE VIEW [dbo].[vwSourceData]
AS
SELECT     Id, Data + ISNULL(Data1, '') AS Data
FROM       dbo.SourceData
GO
```

Запуская:

```
tablediff.exe -sourceserver user14 -sourcedatabase test -sourcetable vwsourcedata -
destinationserver user14 -destinationdatabase test -destinationtable destantiondata –f -c
```

Получаем:

```
ourcedata -destinationserver user14 -destinationdatabase test -destinationt14:19
destantiondata -f -c
Microsoft (R) SQL Server Replication Diff Tool
Copyright (C) 1988-2005 Microsoft Corporation. All rights reserved.

User-specified agent parameter values:
-sourceserver user14
-sourcedatabase test
-sourcetable vwsourcedata
-destinationserver user14
-destinationdatabase test
-destinationtable destantiondata
-f
-c

Table [test].[dbo].[vwsourcedata] on user14 and Table [test].[dbo].[destantionda
ta] on user14 have 2 differences.
Fix SQL written to DIFFIX.633132803438163548.sql.
Err     Id      Col
Src. Only       1
Dest. Only      3
The requested operation took 0,312492 seconds.

D:\Send>
1Help  2UserMn 3View  4Edit  5Copy  6RenMov 7MkFold 8Delete 9ConfMn 10Quit
```

В результате мы видим, что изменения типа **Mismatch** теперь нет, так как значение в записи-источнике (а это у нас уже представление, а не таблица) по полю **Data** совпадает со значением в целевой таблице

Теперь рассмотрим вариант с использованием INSTEAD OF:

Изменим структуру и данные:

```
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO

DELETE FROM [DestantionData]
ALTER TABLE dbo. DestantionData DROP COLUMN Data
GO
ALTER TABLE dbo. DestantionData ADD Data int NOT NULL
GO

IF EXISTS (SELECT * FROM sys.objects WHERE object_id = OBJECT_ID(N'[dbo].[
DestantionRef]') AND type in (N'U'))
    DROP TABLE [dbo].[ DestantionRef]
GO
```

```
CREATE TABLE [dbo].[DestantionRef](
    [Id] [int] IDENTITY(1,1) NOT NULL,
    [Data] [varchar](50) COLLATE Cyrillic_General_CI_AS NOT NULL
 CONSTRAINT [PK_ DestantionRef] PRIMARY KEY CLUSTERED
(
    [Id] ASC
)WITH (IGNORE_DUP_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
GO

CREATE VIEW [dbo].[vwDestantionData]
AS
SELECT dbo.DestantionData.Id, dbo.DestantionRef.Data
FROM       dbo.DestantionData INNER JOIN
dbo.DestantionRef ON dbo.DestantionRef.Id = dbo.DestantionData.Data
GO

INSERT INTO [DestantionRef] (Id, Data) values (1, 'Данные3')
INSERT INTO [DestantionData] (Id, Data) values (2, 1)
```

Запуская:

```
tablediff.exe -sourceserver user14 -sourcedatabase test -sourcetable vwsourcedata -
destinationserver user14 -destinationdatabase test -destinationtable vwdestantiondata -f -c
```

Получаем:

```
ourcedata -destinationserver user14 -destinationdatabase test -destinationt15:07
vwdestantiondata -f -c
Microsoft (R) SQL Server Replication Diff Tool
Copyright (C) 1988-2005 Microsoft Corporation. All rights reserved.

User-specified agent parameter values:
-sourceserver user14
-sourcedatabase test
-sourcetable vwsourcedata
-destinationserver user14
-destinationdatabase test
-destinationtable vwdestantiondata
-f
-c

Table [test].[dbo].[vwsourcedata] on user14 and Table [test].[dbo].[vwdestantion
data] on user14 have 2 differences.
Fix SQL written to DIFFIX.633132832147871828.sql.
Err     Id      Col
Src. Only       1
Mismatch        2       Data
The requested operation took 1,1874544 seconds.

D:\Send>_
1Help  2UserMn 3View  4Edit  5Copy  6RenMov 7MkFold 8Delete 9ConfMn 10Quit
```

В файле:

```
-- Host: user14
-- Database: [test]
-- Table: [dbo].[vwdestantiondata]
INSERT INTO [dbo].[vwdestantiondata] ([Data],[Id]) VALUES ('Данные1',1)
UPDATE [dbo].[vwdestantiondata] SET [Data]='Данные1' WHERE [Id] = 2
```

Для корректного выполнения операций создадим соответствующие триггеры, например на добавление:

```
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO

CREATE TRIGGER [trgDestantionDataInsert]
  ON  [dbo].[vwDestantionData]
  INSTEAD OF INSERT
AS
BEGIN
    SET NOCOUNT ON;
    INSERT INTO dbo.DestantionRef (Data)
    SELECT Data FROM INSERTED WHERE NOT EXISTS
        (SELECT NULL FROM dbo.DestantionRef df WHERE INSERTED.Data = df.Data)

    INSERT INTO dbo.DestantionData (Id, Data)
    SELECT INSERTED.Id, dbo.DestantionRef.Id FROM INSERTED INNER JOIN
dbo.DestantionRef
        ON INSERTED.Data = dbo.DestantionRef.Data
END
GO

CREATE TRIGGER [trgDestantionDataUpdate]
  ON  [dbo].[vwDestantionData]
  INSTEAD OF UPDATE
AS
BEGIN
    SET NOCOUNT ON;
    INSERT INTO dbo.DestantionRef (Data)
    SELECT Data FROM INSERTED WHERE NOT EXISTS
        (SELECT NULL FROM dbo.DestantionRef df WHERE INSERTED.Data = df.Data)

    UPDATE dbo.DestantionData
    SET Data = dbo.DestantionRef.Id
    FROM INSERTED INNER JOIN dbo.DestantionRef ON INSERTED.Data =
dbo.DestantionRef.Data
    WHERE dbo.DestantionData.Id = INSERTED.Id
END
GO

CREATE TRIGGER [trgDestantionDataDelete]
  ON  [dbo].[vwDestantionData]
  INSTEAD OF DELETE
AS
BEGIN
    SET NOCOUNT ON;

    DELETE dbo.DestantionData
    FROM DELETED
    WHERE dbo.DestantionData.Id = DELETED.Id
```

```
END
GO
```

Посмотрим работу утилиты с уникальным индексом на столбцах, в которых допустимы значения NULL:

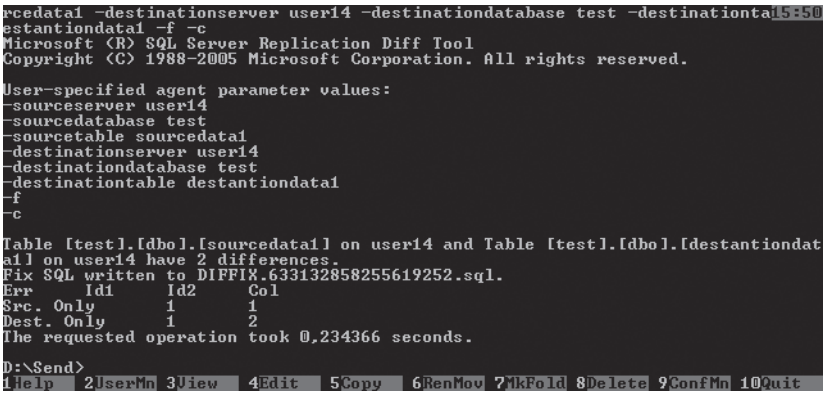
Создание объектов базы данных и заполнение их данными:

```
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
SET ANSI_PADDING ON
GO
CREATE TABLE [dbo].[SourceData1](
    [Id1] [int] NOT NULL,
    [Id2] [int] NULL,
    [Data] [varchar](50) COLLATE Cyrillic_General_CI_AS NOT NULL
) ON [PRIMARY]

GO
SET ANSI_PADDING OFF

CREATE UNIQUE NONCLUSTERED INDEX [IX_SourceData1] ON [dbo].[SourceData1]
(
    [Id1] ASC,
    [Id2] ASC
)WITH (SORT_IN_TEMPDB = OFF, DROP_EXISTING = OFF, IGNORE_DUP_KEY = OFF,
ONLINE = OFF) ON [PRIMARY]

CREATE TABLE [dbo].[DestantionData1](
    [Id1] [int] NOT NULL,
    [Id2] [int] NULL,
    [Data] [varchar](50) COLLATE Cyrillic_General_CI_AS NOT NULL
) ON [PRIMARY]

GO
SET ANSI_PADDING OFF

CREATE UNIQUE NONCLUSTERED INDEX [IX_DestantionData1] ON [dbo].[SourceData1]
(
    [Id1] ASC,
    [Id2] ASC
)WITH (SORT_IN_TEMPDB = OFF, DROP_EXISTING = OFF, IGNORE_DUP_KEY = OFF,
ONLINE = OFF) ON [PRIMARY]

INSERT INTO [SourceData1] (Id1, Id2, Data) values (1, 1, 'Данные1')
INSERT INTO [SourceData1] (Id1, Id2,Data) values (1, NULL, 'Данные2')
INSERT INTO [DestantionData1] (Id1, Id2, Data) values (1, 2, 'Данные1')
INSERT INTO [DestantionData1] (Id1, Id2, Data) values (1, NULL, 'Данные2')
```

Запуская:

```
tablediff.exe -sourceserver user14 -sourcedatabase test -sourcetable sourcedata1 -
destinationserver user14 -destinationdatabase test -destinationtable destantiondata1 -f –c
```

Получаем:



В файле:

– Host: user14
– Database: [test]
– Table: [dbo].[destantiondata1]

```
INSERT INTO [dbo].[destantiondata1] ([Data],[Id1],[Id2]) VALUES ('Данные1',1,1)
DELETE FROM [dbo].[destantiondata1] WHERE [Id1] = 1 AND [Id2] = 2
```

Из чего можно сделать вывод, что сравнение двух значений NULL дало положительный результат (хотя при проверке условия FK сервер не замечает наличия в подчиненной таблице записей, если в каком-либо поле FK содержится значение NULL, разрешая, например удалить запись из главной таблицы).

Хотя программа **TableDiff** не может использоваться для обращения к серверам, отличным от SQL Server, мы можем организовать синхронизацию данных, используя механизм присоединения других источников данных.

Например, в случае публикации данных из базы данных сервера SQL Server в базу данных приложения MS Access, потребуется выполнить следующие действия.

1. Создать присоединенный сервер, в качестве источника указав базу данных приложения MS Access (см. sp_addlinkedserver).
2. Создать представления (в базе данных SQL Server, которую мы используем как источник или новую базу данных) на таблицы базы данных приложения MS Access.
3. Создать триггеры INSTEAD OF на представления.

После этого для синхронизации данных мы сможем применять сценарии обновления, формируемые программой **TableDiff**.

Заключение

Хотя на основе программы **TableDiff** можно построить полный механизм репликации данных, все же эта утилита в основном предназначена для первичной синхронизации данных, так как постоянное полное сравнение данных при больших объемах не всегда является оправданной нагрузкой.

Репликация DDL в SQL Server 2000 и 2005

По материалам статьи Динеш Азанка (Dinesh Asanka) «DDL Replication in SQL Server 2000 and 2005»

Перевод Григория Петровича Корнилова

Введение

Вам может быть интересно, как добавить, удалить или изменить (тип или размер) столбец таблицы, которая является частью публикации. Я видел этот вопрос на многих форумах и дискуссионных досках. Очевидно, что метод, состоящий из удаления репликации, внесения необходимых изменений и затем пересоздания репликации, не эффективен. Ваши пользователи даже могут не позволить вам удалить репликацию, так как этот метод, фактически, приостановит работоспособность системы. В современной обстановке высокой конкуренции вы не можете позволить себе приостанавливать работоспособность системы даже на несколько минут. Поэтому, вам необходимо применять решение, которое позволит вам провести операцию без нарушения процесса репликации.

Реализация в SQL Server 2000

Если вам необходимо модифицировать структуру таблицы в базе данных SQL Server 2000, которая участвует в репликации, вы не можете выполнить стандартные ALTER TABLE операции для добавления, удаления или изменения столбцов. Если вы все же попытаетесь, то это приведет к ошибке, и вы получите примерно следующее сообщение:

```
Unable to modify table.
ODBC error: [Microsoft][ODBC SQL Server Driver][SQL Server] Cannot add columns to
table "Categories" because it is being published for merge replication.
```

Примечание. Вы можете добавить столбец, используя операцию ALTER TABLE ADD в репликации моментальных снимков или в репликации транзакций, однако в итоге это изменение подписчикам передано не будет.

В SQL Server 2000 есть две системные хранимые процедуры sp_repladdcolumn и sp_repldropcolumn, которые вы можете использо-

вать для изменений схем реплицируемых таблиц, и эти изменения в итоге будут воспроизведены на подписчиках.

Если вы хотите добавить столбец, то выполните примерно такую операцию на издателе:

```
sp_repladdcolumn @source_object = N'Customers', @column = N'Cat1', @typetext =
,N'varchar(25)', @publication_to_add = N'data',@from_agent = 0,
@schema_change_script = NULL, @force_invalidate_snapshot = 1 ,
@force_reinit_subscription = 1
```

В результате исполнения этого сценария, будет добавлен новый столбец Cat1 с типом varchar(25) в таблице Customers базы данных издателя, и это действие реплицируется всем подписчикам.

```
sp_repldropcolumn @source_object = 'Customers' , @column = 'Cat1' ,
@schema_change_script = NULL , @force_invalidate_snapshot = 1 ,
@force_reinit_subscription = 1
```

Представленная выше операция удалит столбец Cat1 из таблицы Customers базы данных издателя, и это так же реплицируется всем подписчикам.

Однако проблематично изменить столбец, так как нет хранимой процедуры, реализующей подобную операцию. Тем не менее, комбинируя указанные выше две хранимые процедуры, вы можете достичь этой цели.

```
CREATE TABLE #TEMP (ID VARCHAR(15), DES VARCHAR(100))
go
INSERT INTO #TEMP Select CustomerID,Fax from Customers – Data will be stored in a
temp table
go
sp_repldropcolumn @source_object = 'Customers'
    , @column = 'Fax'
    , @schema_change_script = NULL
    , @force_invalidate_snapshot = 1
    , @force_reinit_subscription = 1
Go

sp_repladdcolumn @source_object = N'Customers', @column = N'Fax', @typetext =
,N'varchar(25)', @publication_to_add = N'data',@from_agent = 0,
@schema_change_script = NULL, @force_invalidate_snapshot = 1 ,
@force_reinit_subscription = 1
go
Update Customers Set Fax = (select DES from #TEMP where Id = Customers.CustomerID)
– Data will be restored from the temp table
drop TABLE #TEMP
```

Показанный выше сценарий изменяет тип столбца Fax на varchar(25), и ранее введенные данные будут добавлены позже. Эти изменения будут реплицированы соответственно всем подписчикам. Однако позиция столбца в таблице будет изменена. После выполнения представленного выше сценария, столбец Fax будет последним в таблице Customers.

Во всех трех случаях вы выполняете сценарий только на издателе, а изменения, включая изменения схемы, будут реплицированы всем подписчикам.

Реализация в SQL Server 2005

В SQL Server 2005 бесспорно многие вещи можно сделать более просто. По-умолчанию, все DDL изменения реплицируются всем подписчикам. Таким образом, уменьшая число потенциальных проблем для разработчиков.

Вы можете легко найти эту опцию в разделе «Subscription Options» диалогового окна свойств издателя. Эта опция называется «Replicate schema changes», которая установлена в значение «True» по умолчанию для всех публикаций.

Важно помнить, что все изменения схем будут выполнены во всех использующих изменяемый объект публикациях.

Репликация между разными версиями

Репликация слиянием может быть реализована между версиями SQL Server 2005 и SQL Server 2000. В смешанном случае, операции ALTER TABLE **не будут** отражены на подписчиках, но операции с использованием системных хранимых процедур sp_repladdcolumn и sp_repldropcolumn подписчикам реплицироваться будут.

Заключение

Репликация изменений схем в SQL Server 2005 реализована по умолчанию, и она удобно управляется средствами подсистем репликации SQL Server.

Репликация программируемых объектов БД в SQL Server 2005

По материалам статьи Байя Павлиашвили (Baya Pavliashvili) «Replicating Code Modules with SQL Server 2005»

Перевод Ирины Николаевны Наумовой

В предыдущих статьях из этой серии рассказывалось о том, как организовать в SQL Server 2005 репликацию статей представляющих собой таблицы. Как и предыдущие версии, SQL Server 2005 предоставляет возможность репликации модулей кода: хранимые процедуры, представления (включая индексированные представления) и пользовательские функции (UDF). В этой статье дается краткий обзор репликации программируемых объектов и даются рекомендации по использованию репликации таких модулей.

Настройка репликации программируемых объектов

Репликация программируемых объектов настраивается также как и репликация таблиц. Публикация в качестве статей может содержать таблицы, индексированные представления, представления, пользовательские функции и хранимые процедуры. Причины, по которым программируемые объекты могут быть добавлены в репликацию, приведены в таблице ниже.

Тип статьи	Причина
Представление	Таблицы, на которых базируется представление, должны существовать на подписчике. Однако эти таблицы могут не участвовать в репликации.
Индексированное представление	Таблицы, на которых базируется представление, должны существовать на подписчике. Однако эти таблицы не участвуют в репликации. На серверах – подписчиках должна быть установлена версия SQL Server 2000 и выше. Все подписчики должны использовать SQL Server в редакции Enterprise Edition.

Тип статьи	Причина
Хранимые процедуры, определяемые пользователем функции	Все объекты, упомянутые в хранимой процедуре или пользовательской функции должны существовать на подписчике. Однако, эти объекты могут не участвовать в репликации.

Для добавления в репликацию представлений, пользовательских функций и хранимых процедур можно воспользоваться мастером создания публикации. После того как вы откроете публикацию нужного типа для издаваемой базы данных, можно приступить непосредственно к работе со статьей. На следующем рисунке продемонстрировано добавление к публикации представления, индексированного представления, хранимой процедуры и пользовательской функции.

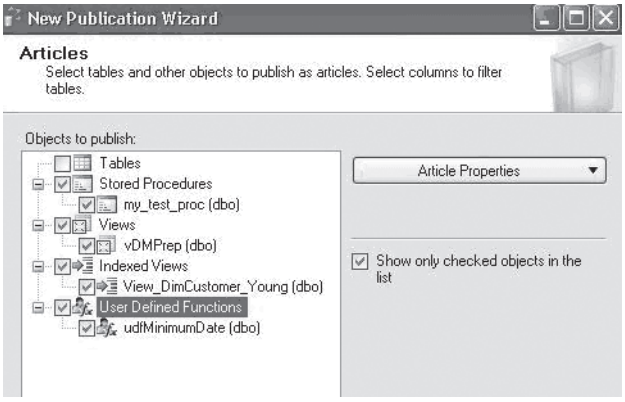


Рис.1.

Чтобы выбрать опции для каждой добавляемой в публикацию статей, нужно нажать кнопку Article Properties. Можно выбрать несколько опций для каждого типа реплицируемых программируемых объектов. Также можно реплицировать схему представлений, индексируемых представлений и пользовательских функций. Для хранимых процедур предусмотрена дополнительная гибкость – помимо их определения вы можете реплицировать и их выполнение. В таблицу ниже сведены опции, доступные для настройки при репликации программируемых объектов.

Тип статьи	Опция/значение	Описание
Представления	Copy User Triggers: True or False	Для реплицируемого представления создается триггер в базе данных подписчика, если он есть на издателе.
Представления, индексированные представления, хранимые процедуры, определяемые пользователем функции.	Copy Extended Properties: True or False	На подписчике создаются расширенные свойства реплицируемого представления.

Тип статьи	Опция/значение	Описание
Представления, индексированные представления, хранимые процедуры, определяемые пользователем функции.	Destination Object Name/Destination Object Owner	Вы можете создать реплицируемый объект с тем же именем что и на издателе и владельцем объекта или с другим именем и/или владельцем.
Представления, индексированные представления, хранимые процедуры, определяемые пользователем функции.User Defined Function	Action if name in use: Keep existing object unchanged OR Drop existing object and create a new one	Запомните, если сохраняете существующий объект, определение этого программируемого модуля на издателе и подписчике может быть разным.
Представления, индексированные представления, хранимые процедуры, определяемые пользователем функции.	Create Schemas at Subscriber: True or False	Определяет, должен ли выполняться оператор CREATE SCHEMA на подписчике, если там нет схемы объекта.
Хранимые процедуры	Replicate: Stored procedure definition only; Execution of the stored procedure; Execution in a serialized transaction of the SP.	Определяет, должно ли реплицироваться выполнение хранимой процедуры. Запомните, оператор ALTER PROCEDURE будет реплицировать изменения схемы даже для публикаций, в которых реплицируется выполнение хранимой процедуры, таким образом, изменения в определении хранимой процедуры будут всегда доставлены подписчику (если только вы явно не задали репликацию изменений схемы).

Как только Вы установили свойства для каждой статьи, можно создать снимок для публикации немедленно и/или создать расписание для запуска Snapshot Agent. Следующим шагом необходимо определить параметры настройки безопасности для Snapshot Agent и Log Reader agent, проанализируйте то, что должен сделать мастер, и нажмите кнопку Finish, чтобы создать публикацию.

Мастер создания публикации предоставит данные о каждом типе программируемых объектов, которые вы пытаетесь реплицировать. Однако, публикация может быть создана даже если на подписчике нет объектов, на которые она использует.

Процесс создания подписки для публикации программируемых объектов идентичен процессу репликации табличных статей. Если любой из объектов, на которые ссылается копируемый модуль кода, отсутствует на подписчике, агент распределения уведомит вас об ошибке, но подписка будет создана. Сообщения об ошибках в работе агента распределения помогают понять причины проблем, потому что с помощью этих сведений можно определить закончившуюся ошибкой команду (см. рис.)).

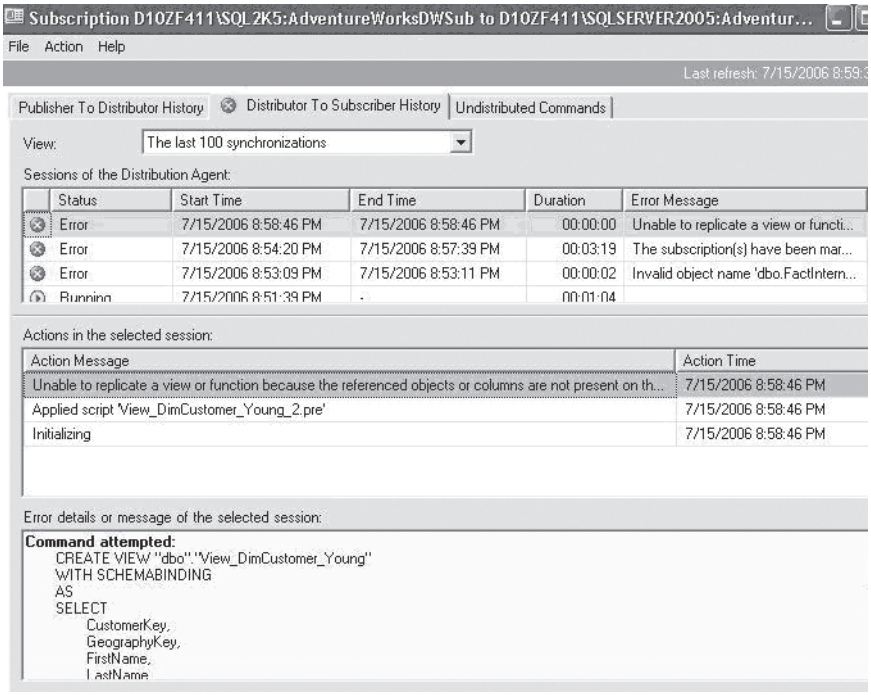


Рис.2.

Примечание. Ошибки, возникающие при репликации программируемых объектов, обычно легко обнаружимы. Таких ошибок можно избежать, если внимательно читать экраны мастера и выполнить все требования.

Следующий сценарий создает публикацию, в которую входят хранимая процедура, представление, индексированное представление и пользовательская функция:

```
exec sp_addpublication @publication = n'pub_name',
@description=N'Transactional publication of database ''AdventureWorksDW'' .',
@sync_method = N'concurrent', @retention = 0, @allow_push = N'true',
@allow_pull = N'true', @allow_anonymous = N'true',
@enabled_for_internet = N'false', @snapshot_in_defaultfolder = N'true',
@compress_snapshot = N'false', @ftp_port = 21, @ftp_login = N'anonymous',
@allow_subscription_copy = N'false', @add_to_active_directory = N'false',
@repl_freq = N'continuous', @status = N'active',
@independent_agent = N'true', @immediate_sync = N'true',
@allow_sync_tran = N'false', @autogen_sync_procs = N'false',
@allow_queued_tran = N'false', @allow_dts = N'false', @replicate_ddl = 1,
@allow_initialize_from_backup = N'false', @enabled_for_p2p = N'false',
@enabled_for_het_sub = N'false'
GO
```

```
--Добавление статьей в публикацию репликации транзакций
--Определяемая пользователем функция:
exec sp_addarticle @publication = n'pub_name',
@article = N'udfMinimumDate', @source_owner = N'dbo',
@source_object = N'udfMinimumDate', @type = N'func schema only',
@description = N'', @creation_script = N'', @pre_creation_cmd = N'drop',
@schema_option = 0x0000000008000001, @destination_table = N'udfMinimumDate',
@destination_owner = N'dbo', @status = 16
GO

--Хранимая процедура:
exec sp_addarticle @publication = n'pub_name',
@article = N'update_factFinance', @source_owner = N'dbo',
@source_object = N'update_factFinance', @type = N'proc exec',
@description = N'', @creation_script = N'', @pre_creation_cmd = N'drop',
@schema_option = 0x0000000008000001, @destination_table = N'update_factFinance',
@destination_owner = N'dbo', @status = 0
GO

--Индексированное представление:
exec sp_addarticle @publication = n'pub_name',
@article = N'View_DimCustomer_Young', @source_owner = N'dbo',
@source_object = N'View_DimCustomer_Young', @type = N'indexed view schema only',
@description = N'', @creation_script = N'', @pre_creation_cmd = N'drop',
@schema_option = 0x0000000008000001, @destination_table =
N'View_DimCustomer_Young',
@destination_owner = N'dbo', @status = 16
GO

--Представление:
exec sp_addarticle @publication = n'pub_name',
@article = N'vTimeSeries', @source_owner = N'dbo',
@source_object = N'vTimeSeries', @type = N'view schema only', @description = N'',
@creation_script = N'', @pre_creation_cmd = N'drop',
@schema_option = 0x0000000008000001, @destination_table = N'vTimeSeries',
@destination_owner = N'dbo', @status = 16
GO
```

Изменение схемы репликации

Вспомните, в предыдущей версии SQL Server для того, чтобы определения программируемых объектов передались подписчику, нужно было запустить агента создания снимка. В SQL Server 2005 это уже не так: репликация передает операторы ALTER VIEW, ALTER FUNCTION, ALTER PROCEDURE и ALTER TRIGGER подписчику в реальном времени. Опция репликации триггеров уже не является единственной возможностью для статей соответствующих типов, но она по-прежнему позволяет копировать триггеры, определенные на таблице или

представлении в публикуемой базе данных. Запомните, что нельзя реплицировать DDL триггеры (триггеры языка определения данных).

Давайте рассмотрим репликацию изменения индексируемого представления. Я создал очень простой пример индексируемого представления на издателе с помощью следующих команд:

```
CREATE VIEW [dbo].[View_DimCustomer_Young]
WITH SCHEMABINDING
AS
SELECT CustomerKey, GeographyKey, FirstName, LastName, BirthDate
FROM  dbo.DimCustomer
WHERE (BirthDate > CONVERT(SMALLDATETIME, '1/1/1980', 101))
GO

CREATE UNIQUE CLUSTERED INDEX [ix_DCY_CustomerKey] ON
[dbo].[View_DimCustomer_Young]
(
    [CustomerKey] ASC
)WITH (PAD_INDEX = OFF, SORT_IN_TEMPDB = OFF, DROP_EXISTING = OFF,
IGNORE_DUP_KEY = OFF, ONLINE = OFF)
ON [PRIMARY]
```

Это представление возвращает несколько столбцов таблицы DimCustomer для записей клиентов, которые родились после 1 января 1980 года. Я добавил это представление в публикацию репликации транзакций и создал для нее подписку на другом сервере.

Теперь давайте изменим представление, чтобы оно возвращало клиентов, родившихся после 1 января 1978 года. Для этого я выполнил следующий код:

```
ALTER VIEW [dbo].[View_DimCustomer_Young]
WITH SCHEMABINDING
AS
SELECT CustomerKey, GeographyKey, FirstName, LastName, BirthDate
FROM  dbo.DimCustomer
WHERE (BirthDate > CONVERT(SMALLDATETIME, '1/1/1978', 101))
```

Теперь, если я в контексте базы данных распространителя выполняю системную хранимую процедуру sp_browserplcmds, я найду там команду ALTER VIEW, которая предназначена для передачи подписчику.

Репликация программируемых модулей особенно полезна, если Вы работаете в мультисерверной среде и распределяете прикладную нагрузку на несколько серверов с идентичными представлениями, пользовательскими функциями и хранимыми процедурами. Вместо того, чтобы применить изменения схемы на нескольких серверах, можно просто выполнить эти изменения на одном сервере-издателе, после чего эти изменения будут растиражированы для всех подписчиков. В предыдущих версиях SQL Server нужно было запустить Snapshot агента, который бы создал моментальный снимок, с помощью которого изменения схемы до-

ставляются подписчикам. В версии SQL Server 2005 изменения схемы будут доставляться также как изменения данных. Это позволяет значительно упростить и ускорить задачи развертывания приложений.

Иногда может понадобиться, чтобы изменения схемы не передавались пока не отработает Snapshot агент. Например, для того чтобы проверить как передаются изменения хранимой процедуры на один из серверов перед развертыванием этих изменений на все промышленные сервера, вы можете отключить передачу изменений схемы в опциях подписчика диалогового окна свойств публикации, показанного ниже.

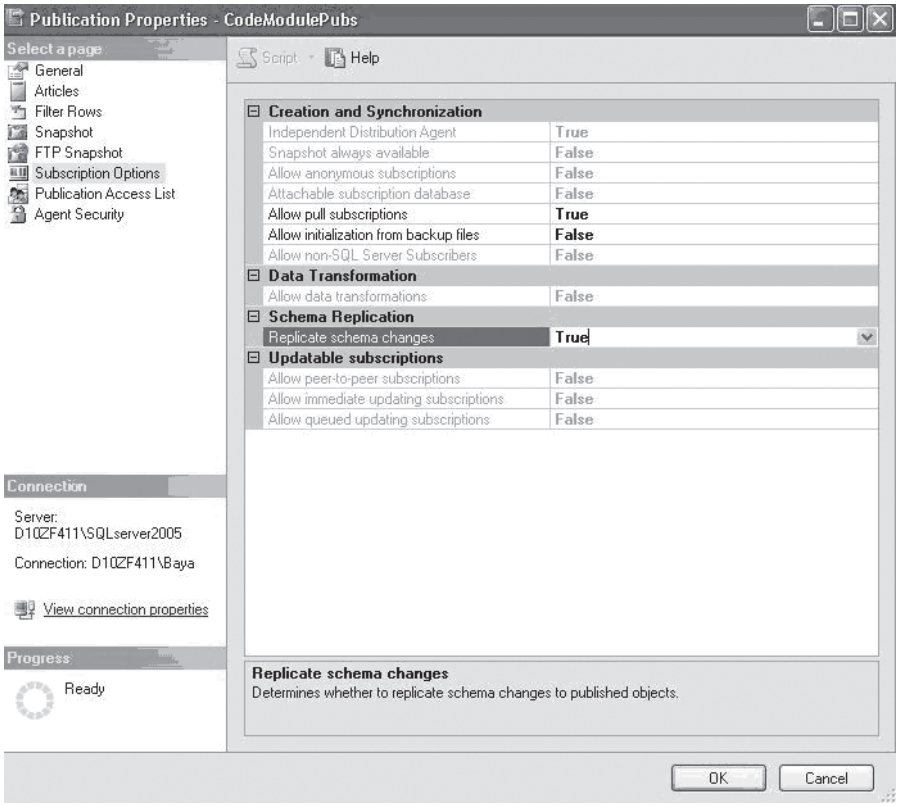


Рис. 3.

Репликация индексируемых представлений как таблиц

Репликация индексированных представлений осуществляется также как репликация таблиц; в этом случае, SQL Server создает таблицу на подписчике, которая содержит те же данные, что и индексированное представление на издателе. Данные изменяются в индексированном представлении на издателе и передаются в таблицу на подписчике. Обратите внимание, что таблица, на которой основано индексированное представление, не обязана существовать на подписчике.

Репликация индексированных представлений как таблиц может быть полезна, когда на подписчике необходимо только лишь подмножество данных таблицы. Например, если на подписчике нужна информация только о клиентах, дата рождения которых больше 1.1.1998 , можно настроить репликацию индексированного представления, приведенного ниже, используя его в качестве статьи, вместо того чтобы копировать всю таблицу dimCustomer.

Для того чтобы выполнять репликацию индексированного представления как таблицы, нужно изменить параметр @type системной процедуры sp_addarticle. По умолчанию этот параметр установлен в значение N'indexed view logbased'. Например, следующий код добавит индексированное представление View_DimCustomer_Young в существующую публикацию как таблицу.

```
exec sp_addarticle
@publication = N'publication_name',
@article = N'View_DimCustomer_Young',
@source_owner = N'dbo',
@source_object = N'View_DimCustomer_Young',
@type = N'indexed view logbased',
@description = null,
@creation_script = null,
@pre_creation_cmd = N'none',
@schema_option = 0x0000000008000001,
/* table name doesn't have to be the same as view name */
@destination_table = N'View_DimCustomer_Young',
@destination_owner = N'dbo'
GO
```

После настройки репликации индексированного представления как таблицы, операторы INSERT, UPDATE и DELETE, выполненные на представлении на издателе, будут реплицированы в таблицу на подписчике.

Репликация выполнения хранимых процедур

Таким же образом можно настроить репликацию выполнения хранимых процедур, что очень полезно при больших изменениях в имеющихся данных, и при условии, что данные на подписчике и издателе идентичны. Что произойдет, если выполнение оператора UPDATE затрагивает 1000 строк реплицируемой таблицы? По умолчанию SQL Server трансформирует одну команду UPDATE в выполнение хранимой процедуры репликации 1000 раз. Этот вариант хорош тем что каждое выполнение хранимой процедуры репликации затрагивает только одну строку что не вызывает большое количество блокировок/подтверждений на подписчике.

Но что произойдет, если ваша хранимая процедура выполняет изменения, затрагивающие миллион строк в нескольких таблицах? Ваша база данных распределения будет расти экспоненциально, и время задержки репликации может стать недопустимо большим. Перед тем как пере-

дать эти команды подписчику SQL Server должен прочитать их из таблицы msrepl_commands базы данных распределения; Агент - чистильщик распределителя занимается удалением транзакции для этих таблиц, когда они уже были переданы подписчику. Если таблица msrepl_commands содержит несколько миллионов строк, чтение и удаление данных из этой таблицы будет выполняться очень медленно. Кроме того, передача больших изменений при использовании табличной статьи, оказывает большую нагрузку на сеть.

Репликация выполнения хранимых процедур предлагает более эффективную альтернативу стандартному методу, потому что при этом выполняется одна и та же хранимая процедура, и на издателе и подписчике, уменьшая, таким образом, сетевой трафик и число команд в базе распределения. Если нужно передать изменения 50 миллионов строк, и известно, что издатель и подписчик идентичны, было бы более эффективно включать в репликацию выполнение хранимой процедуры, которая осуществляет эти изменения.

Другой альтернативой при репликации больших изменений одной таблицы является репликация таблицы путем выполнения одинакового оператора UPDATE на издателе и подписчике (Загляните в первую статью из этой серии, если Вы хотите узнать как это можно сделать). Репликация выполнения хранимой процедуры может быть лучшим выбором, однако, это справедливо обычно для массовых изменений в нескольких таблицах.

Примечание. Если вы выполняете хранимую процедуру внутри явной транзакции, эта транзакция должна быть завершена перед тем как процедура будет реплицирована.

Например, предположим что у нас есть хранимая процедура, которая изменяет некоторое количество строк в таблице factFinance базы данных AdventureWorksDW :

```
CREATE PROC update_factFinance (
 @PercentChange NUMERIC (3,2),
 @OrganizationKey TINYINT,
 @TimeKey INT)
AS

/*
Изменяем количество выданных ключей
*/
UPDATE factFinance
SET amount = amount * @PercentChange
WHERE OrganizationKey = @OrganizationKey
AND TimeKey = @TimeKey
```

Создадим публикацию на основе репликации выполнения хранимой процедуры. Каждый раз, когда мы выполняем процедуру на издателе, агент распределения будет передавать подписчику команду, подобную этой:

```
{call "dbo"."update_factFinance " (1.10, 3, 32)}
```

Запомните, что репликация будет просто передавать эту команду, репликация не будет проверять, затрагивает ли выполнение команды какие-либо строки на издателе и подписчике. Таким образом, для того чтобы обеспечить целостность данных на издателе и подписчике, необходимо перед использованием репликации выполнения хранимой процедуры удостовериться в том, что данные на подписчике и издателе идентичны.

Репликация выполнения хранимых процедур внутри сериализуемой транзакции

Хранимую процедуру можно включить в репликацию в качестве статьи, только если она выполняется внутри сериализуемой транзакции. Это требует соблюдения двух условий, обеспечить которые необходимо перед тиражированием процедуры подписчику:

- Уровень изоляции транзакций у подключения, в котором выполняется хранимая процедура, должен быть установлен в SERIALIZABLE.
- Необходимо выполнять процедуру внутри явной транзакции, используя операторы BEGIN TRANSACTION / COMMIT TRANSACTION.

Если хотя бы одно условие не будет выполнено, выполнение хранимой процедуры не реплицируется. Помимо этих обязательных условий, также необходимо использовать опцию SET XACT_ABORT ON. Использование этой опции гарантирует, что изменения, внесенные транзакцией, внутри которой выполняется хранимая процедура, будут автоматически отменены, если возникнут ошибки времени выполнения.

Репликация выполнения хранимых процедур внутри сериализуемой транзакции – это рекомендуемая опция, когда необходимо поддержать целостность данных на издателе и подписчике. Почему? В каждой хранимой процедуре содержится несколько явных или неявных транзакций. Вы можете столкнуться с ситуацией когда некоторые транзакции внутри хранимой процедуры завершаются успешно, а другие с ошибкой. Если Вы заставляете SQL Server реплицировать каждое выполнение хранимой процедуры, тогда даже то выполнение, в котором транзакции завершаются ошибками, будет отправлено подписчику. Уровень изоляции транзакций – SERIALIZABLE, является самым жестким уровнем изоляции, гарантирующим, что блокировки будут установлены на всех таблицах, которые использует хранимая процедура. Блокировки будут удерживаться до тех пор, пока транзакция не будет завершена. Поэтому,

использование в репликации выполнение только в пределах сериализуемой транзакции, дает гарантию того, что процедура успешно завершит работу на издателе, и только потом будет послана подписчикам.

Давайте воспользуемся процедурой update_factFinance чтобы продемонстрировать как мы можем реплицировать ее выполнение внутри сериализуемой транзакции. Выполним следующий код:

```
SET TRANSACTION ISOLATION LEVEL SERIALIZABLE
SET XACT_ABORT ON
BEGIN TRAN
EXEC update_factFinance 1.10, 3, 32

COMMIT
```

Однако, следующая команда не будет реплицирована, потому что она не включена в явную транзакцию:

```
EXEC update_factFinance 1.10, 3, 32
```

Теперь мы завершим выполнение процедуры, чтобы продемонстрировать значение установки XACT_ABORT. Я изменяю тип данных столбца amount таблицы factFinance на SMALLINT, вместо INT, выполняя следующую инструкцию:

```
ALTER TABLE factFinance ALTER COLUMN amount SMALLINT
```

Максимальное значение для типа SMALLINT – 32768; умножаем максимальное значение столбца на 1.15 чтобы результат превысил 32768, таким образом, следующее выполнение процедуры update_factFinance, приведет к ошибке:

```
SET TRANSACTION ISOLATION LEVEL SERIALIZABLE
/* CORRECT setting*/
SET XACT_ABORT ON
BEGIN TRAN
EXEC update_factFinance 1.15, 3, 32

COMMIT
```

Результат таков:

```
Msg 8115, Level 16, State 2, Procedure update_factFinance, Line 10
Arithmetic overflow error converting expression to data type smallint.
```

Транзакция отменена, и выполнение хранимой процедуры не передано подписчику.

Далее, выполним тот же набор команд, отменив установку XACT_ABORT:

```
SET TRANSACTION ISOLATION LEVEL SERIALIZABLE
/* НЕКОРРЕКТНАЯ УСТАНОВКА! Это сделано только в демонстрационных целях! */
SET XACT_ABORT OFF
BEGIN TRAN
```

```
EXEC update_factFinance 1.15, 3, 32
COMMIT
```

Выполнение хранимой процедуры завершится с той же ошибкой что и в первый раз. Однако при проверке базы данных распределителя, мы найдем следующую команду, посланную подписчику:

```
{call "dbo"."update_factFinance" (1.15,3,32)}
```

Выполнение процедуры передано подписчику даже при условии того, что на издателе оно завершилось с ошибкой. Это приведет к тому, что агент распределения завершит работу с ошибкой. И что еще более важно, может нарушить целостность данных на подписчике и издателе. Поэтому всегда используйте опцию SET XACT_ABORT ON при репликации выполнения хранимых процедур.

Вывод

Данная статья рассказывает о том, как выполняется репликация хранимых процедур, представлений и пользовательских функций в SQL Server 2005. По сравнению с другими версиями, возможно, самым важным усовершенствованием является возможность применить изменения на подписчике без запуска агента создания снимка.

Эта серия статей познакомила вас с репликации транзакций в SQL Server 2005. Построенная на твердой основе предыдущих версий, репликации транзакций в SQL Server 2005 – это зрелая технология, которая способна поддержать приложения класса предприятия. Подобно любой технологии, репликация работает хорошо, если используется к месту и решает присущие ей задачи. Убедитесь, что изучили тонкости работы репликации, и постарайтесь очень тщательно спланировать внедрение решений с ее использованием.

Репликация полнотекстовых индексов в SQL Server 2005

По материалам статьи Пола Ибизона (Paul Ibison) «Replicating Full-Text Indexes in SQL Server 2005»

Перевод Яна Дмитриевича Либермана

Введение

Одно из преимуществ SQL Server 2005 это – более глубокий контроль свойств реплицируемых статей. В частности, можно точно указать, какие объекты следует реплицировать в составе статьи. Далее, я исследую возможность репликации полнотекстовых индексов (full-text index), которые могут быть созданы для опубликованных таблиц.

Реализация

Чтобы опробовать эту новую функциональность, я создал простую транзакционную публикацию и добавил в нее две статьи — таблицу и хранимую процедуру. Таблица называется tTest. Ее структура приведена на рисунке 1.

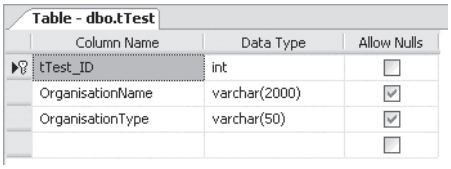


Рис. 1.

Для столбца OrganisationName таблицы tTest создан полнотекстовый индекс. Как я уже писал выше, вместе с таблицей к публикации я добавил хранимую процедуру. Она называется TestFTISearch. Процедура использует полнотекстовый предикат «CONTAINS» для выполнения поиска по данным таблицы tTest. Код процедуры приведен ниже:

```
CREATE Procedure [dbo].[TestFTISearch]
as
  select * from ttest
  where contains (*, 'ltd')
```

После добавления таблицы к публикации, установим значение «True» для свойства статьи «Copy full text indexes» (значение по умолчанию для этого свойства равно «False»). Эта опция определяет, должны ли реплицироваться на подписчика полнотекстовые индексы или нет.

```
CREATE Procedure [dbo].[TestFTISearch]
as
select * from ttest
where  contains (*,'ltd')
```

Рис. 2.

Вполне логично было бы предположить, что мы сделали все что нужно, и настройка на этом закончена.

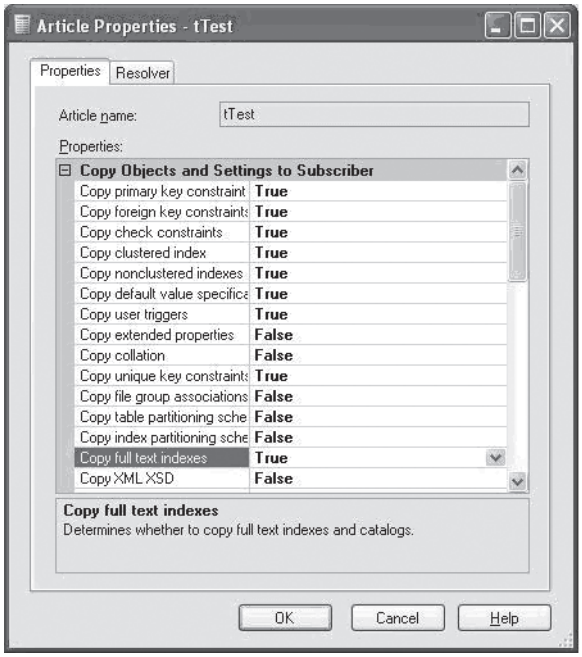


Рис. 3.

Однако, попытка выполнить инициализацию, может привести к следующей фатальной ошибке:

> The schema script 'MyCatalog_4.sch' could not be propagated to the subscriber. (Source: MSSQL_REPL, Error number: MSSQL_REPL-2147201001)
>
> Get help: http://help/MSSQL_REPL-2147201001
>
> Full-Text Search is not enabled for the current database. Use sp_fulltext_database to enable full-text search for the database. The functionality to disable and enable full-text search for a database is deprecated. Please change your application. (Source: MSSQLServer, Error number: 7616)

Причина этой ошибки в том, что полнотекстовое индексирование не было разрешено в базе данных подписки. Чтобы исправить ситуацию мы можем либо «вручную» разрешить полнотекстовое индексирование в базе данных подписки, либо написать сценарий, который будет выполняться до применения моментального снимка:

```
exec sp_fulltext_database 'enable'
```

Примечание. При использовании этой команды нет необходимости проверять текущие настройки базы данных с помощью функции databasepropertyex, так как процедура sp_fulltext_database не выдает ошибку, в случае если полнотекстовый поиск уже был разрешен в базе данных ранее.

После того как мы убедились, что полнотекстовое индексирование включено в базе данных подписки, мы можем столкнуться с еще одной ошибкой:

```
Error messages:

The schema script 'TestFTISearch_3.sch' could not be propagated to the subscriber.
(Source: MSSQL_REPL, Error number: MSSQL_REPL-2147201001)

Get help: http://help/MSSQL_REPL-2147201001

Cannot use a CONTAINS or FREETEXT predicate on table or indexed view “ttest” because
it is not full-text indexed. (Source: MSSQLServer, Error number: 7601)
```

Это может показаться странным, поскольку мы установили опцию реплицировать полнотекстовые индексы и даже проверили файлы моментального снимка и убедились, что поддержка полнотекстовых индексов на табличном уровне действительно была туда включена. Проблема заключается в том, что для репликации транзакций новая публикация, по умолчанию создается с параметром sync_method (режим синхронизации) равным «concurrent». При использовании этого режима синхронизации, ограничения уникальности (unique constraints), от которых зависят полнотекстовые индексы, не создаются до конца так называемой фазы согласования (reconciliation phase). Эта фаза наступает через некоторое время после того, как все файлы моментального снимка будут применены (в процессе фазы согласования соблюдение ограничений уникальности не гарантируется). Так как же преодолеть эту проблему? Мы можем просто изменить значение свойства публикации sync_method на «native» — значение, которое принято по умолчанию в SQL Server 2000. Минусом этого решения является то, что в процессе создания моментального снимка, таблицы на издателе будут заблокированы в течение более длительного времени. Но, по крайней мере, мы достигнем нашей цели. Стоит отметить, что эта проблема никогда не возникает в репликации

слиянием, поскольку в репликации слиянием нет режима синхронизации «concurrent».

```
sp_changepublication @publication = 'yyy'
, @property = 'sync_method'
, @value = 'native'
, @force_invalidate_snapshot = 1
```

При использовании RTM-версии SQL Server 2005, может появиться еще одна ошибка, относящаяся к теме этой статьи (в версии SQL Server 2005 SP1 этой ошибки уже нет). А именно, если в файлах моментального снимка присутствуют вызовы процедур, относящихся к полнотекстовым индексам, но нет обязательного вызова sp_fulltext_column, то это означает, что в полнотекстовых индексах использован язык по умолчанию (default language). В этом случае до момента создания моментального снимка на издателе, нужно явно указать язык.

Выводы

На сегодняшний день в SQL Server 2005 при настройке репликации полнотекстовых индексов в контексте репликации транзакций, может возникнуть ряд описанных выше ошибок. Но используя эту статью как инструкцию, все эти сложности можно легко обойти.

Модель репликации с переизданием

По материалам статьи Пола Ибизона (Paul Ibison) «Replication Republishing»

Перевод Яна Дмитриевича Либермана

Введение

Под переизданием понимается сценарий, в котором издатель посылает данные подписчику, а подписчик, в свою очередь, переиздает данные другим подписчикам. Для этого сценария, в случае репликации транзакций, изменения, произведенные на издателе, будут доставляться конечному подписчику за два шага (double-hop). В репликации слиянием изменения, внесенные в любом месте цепи, будут распространены на все остальные базы данных, включенные в схему. Такая конфигурация, на первый взгляд, может показаться излишне сложной. Так зачем тогда ее использовать? Эта схема может быть очень полезна в случаях, когда группы подписчиков соединены «слабым» или дорогим каналом связи. Используя переиздание, можно добиться того, что данные будут передаваться через «слабое» соединение только один раз, а дальнейшее их распространение будет происходить на противоположной стороне по хорошим каналам связи.

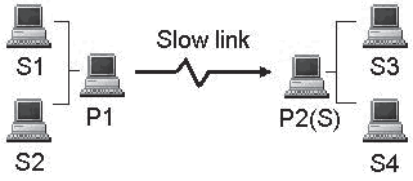


Рис. 1.

Одна из областей, где я пришел к необходимости использовать переиздание — это морские перевозки. Спутниковая связь с кораблями обычно работает медленно и зачастую ненадежно (большие волны мешают приемнику, и связь часто рвется). Поэтому, вместо того чтобы реплицировать данные из головного офиса сразу в несколько баз данных на каждом судне, мы передаем данные только в какую-то одну базу данных на каждом корабле. Дальше, в границах корабля, данные распространяются по быстрой и надежной локальной сети. Эта статья посвящена репликации

с переизданием в SQL Server 2005, однако, то же самое в значительной степени справедливо и для SQL Server 2000.

Настройка

По большей части, настроить репликацию с переизданием не сложно, однако есть ряд важных моментов, на которые следует обратить особое внимание. Для начала я предлагаю рассмотреть общие шаги для репликации слиянием и транзакций, а затем сконцентрироваться на тонкостях, которые относятся исключительно к своему типу репликации.

Инициализация

Если принять за основу схему, приведенную на рисунке 1, то настройка репликации должна выполняться слева направо. То есть, сначала нужно создать публикацию P1, затем создать и инициализировать подписку P2(S). После чего переиздать подписку P2(S). И, наконец, создать и инициализировать подписки S3\4. Все настраивается стандартно и никаких проблем здесь обычно не возникает. Другое дело, если P1 настраивается позже P2(S). Для этого случая мы можем воспользоваться методом резервного копирования и восстановления, но это намного сложнее и, по моему мнению, такого сценария следует избегать. Например, может возникнуть проблема с диапазонами идентификаторов, так как в правильной конфигурации только P1 должен являться первичным координатором диапазонов идентификаторов.

Репликация слиянием

Тип подписки

В SQL Server 2005 существует два типа подписок на репликацию слиянием: клиентская подписка и серверная подписка. Клиентские подписки используют значение приоритета издателя при разрешении конфликтов с другими подписчиками. Для серверной подписки значение приоритета задается явно. В SQL Server 2000 серверной (server) подписке соответствовала глобальная (global) подписка, а клиентской (client) подписке соответствовала локальная подписка (local). Только серверные подписки (которым назначен приоритет) могут переиздавать данные для других подписчиков. Соответственно, в нашей схеме (рисунок 1), для того, чтобы подписку P2(S) можно было переиздать, нужно чтобы это была серверная подписка (с установленным приоритетом). В противном случае будет выдано сообщение об ошибке, приведенное на рисунке 2. Это сообщение по-прежнему использует терминологию SQL Server 2000, хотя смысл понятен.

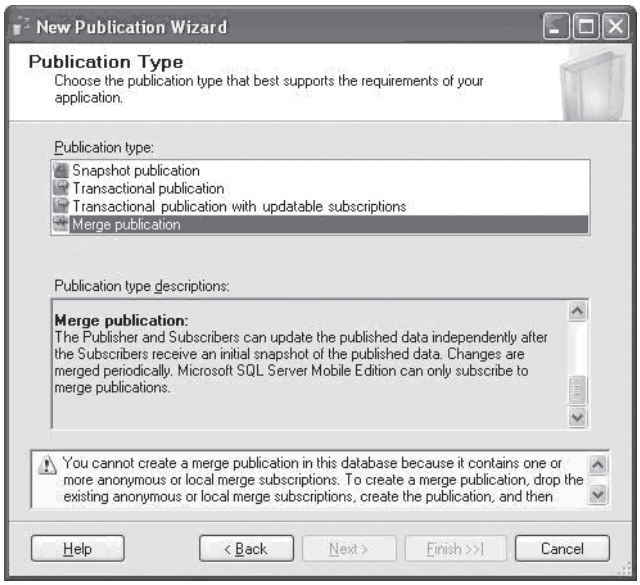


Рис. 2.

Таким образом, подписке P2(S) нужно явно присвоить значение приоритета. Конкретное значение выбирается в соответствии со следующим правилом: для переиздающего подписчика значение приоритета должно быть меньшим, чем у издателя, находящегося выше подписчика. То есть, для нашего случая P1 > P2(S) > S3/4.

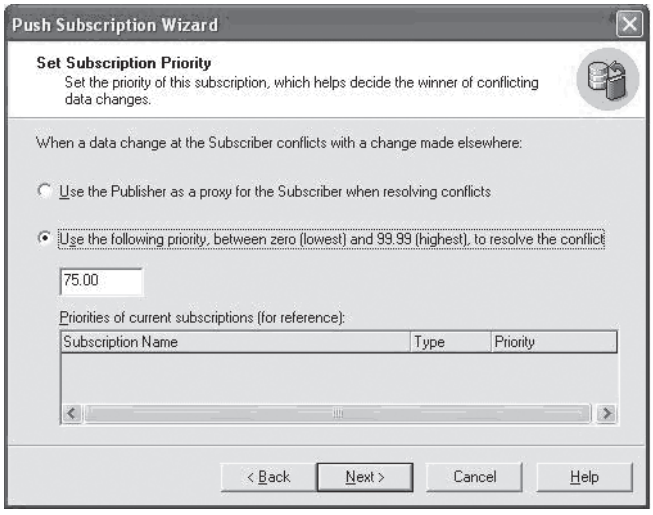


Рис. 3.

Диапазоны идентификаторов

Диапазоны идентификаторов управляются издателем и распространяются подписчикам агентом слияния. В иерархической схеме репликации слиянием с переизданием диапазоны управляются корневым издателем и переиздающими подписчиками. Для нашей схемы, например, если подписка S3 исчерпает свой диапазон идентификаторов, то новый диапазон будет запрошен у P2(S). Когда в свою очередь он тоже закончится, будет запрошен еще один диапазон и так далее. Когда диапазон идентификаторов у P2(S) будет исчерпан, то новый будет запрошен у P1.

Репликация транзакций

Возможности репликации транзакций в схеме с переиздающими подписчиками ограничиваются базовой функциональностью. Например, не поддерживаются обновляемые подписки. Кроме того, трассировочные маркеры (tracer tokens) не пересылаются переиздающими подписчиками. Применительно к нашей схеме это означает, что мы должны добавлять один трассировочный маркер на P1 и другой независимый трассировочный маркер на P2(S).

Изменение схемы, срок хранения публикаций и SQL Server 2000

Изменение схемы для опубликованных объектов должно выполняться только на корневом издателе, а не на переиздающих подписчиках. Изменения, которые распространяются на переиздающих подписчиков, по умолчанию, в свою очередь распространяются и на их подписчиков. В SQL Server 2000 мы использовали процедуры sp_repladdcolumn и sp_repldropcolumn для добавления и удаления столбцов в опубликованных таблицах. Но эти процедуры не поддерживают топологию с переиздающими подписчиками. То есть для нашей схемы (рисунок 1), если бы мы воспользовались процедурой sp_repladdcolumn на P1, то новый столбец был бы добавлен на P1 и P2(S), но не на S3 и S4. Как результат, потребовалась бы повторная инициализация. В SQL Server 2005 для решения той же задачи адаптирована команда ALTER TABLE. В том же сценарии, если добавить столбец командой ALTER TABLE, и при условии, что публикация создавалась с параметром @replicate_ddl = 1, то новый столбец будет добавлен на P2 (S), а затем на S3 и S4. Значение срока хранения публикации (publication retention period) для любого переиздающего подписчика должно быть меньше или равно сроку хранения публикации, установленного на исходном издателе. Использование неправильного значения может привести к несогласованности данных.

Репликация транзакций

Высокая доступность в Репликации SQL Server 2008 с зеркалированием и Доставкой журналов

Автор Александр Юрьевич Гладченко

Статья написана по мотивам технического документа Майкрософт «SQL Server Replication: Providing High Availability using Database Mirroring» и описания в электронной документации SQL Server 2008 Books Online (далее BOL) «Репликация и зеркальное отображение базы данных».

В этой статье мы рассмотрим новые возможности обеспечения высокой доступности тиражируемых данных, используя для этого Репликацию транзакций, Доставку журналов и зеркальные копии баз данных.

Основанная на Репликации транзакций распределенная система хранения данных может обеспечить высокую устойчивость к отказам серверов баз данных. Подобные решения позволяют достичь высокой степени доступности, за счет поддержки избыточных копий данных. Кроме Репликации, избыточность на уровне баз данных способны обеспечить несколько механизмов SQL Server 2008. Это такие возможности, как резервное копирование с последующим восстановлением, Доставка журналов и Зеркальное отображение базы данных. Причем, Зеркальное отображение является единственным механизмом, который поддерживает точную копию защищаемой базы данных практически в реальном масштабе времени, и гарантирует отсутствие потерь данных.

В этой статье на примерах мы посмотрим, как можно использовать Зеркальное отображение реплицируемой базы данных для повышения ее доступности. Мы рассмотрим как Репликация и Зеркальное отображение влияют друг на друга, а также, как Зеркальное отображение совместимо с Доставкой журналов и как Доставка журналов совместима с Репликацией. Кроме того, в этой статье мы коснемся возможностей использования для первоначальной синхронизации баз данных механизмов Доставки журналов, и вкратце рассмотрим принципы работы инициализации подписчика, основанной на логических номерах виртуальных журналов (LSN), которая позволяет сократить время восстановления после отказа при наличии зеркальной копии базы данных Подписчика.

Совместимость Доставки журналов с Зеркальным отображением и Репликацией

В настоящей статье мы не ставим целью полностью раскрыть тему совмещения Доставки журналов с Зеркальным отображением баз данных и Репликацией. Эта тема достаточно подробно раскрыта в BOL. Здесь мы рассмотрим только те аспекты Доставки журналов, которые нам пригодятся для первоначальной синхронизации основной базы данных Зеркального отображения и баз данных Подписчиков в топологии репликации. Коротко будет сказано и о том, как можно использовать Доставку журналов в целях резервного копирования участвующих в Репликации баз данных.

Целью Доставки журналов SQL Server 2008 является обеспечение автоматической синхронизации баз данных, которая осуществляется путем резервного копирования журналов транзакций в базе данных Источника, и последующей доставке и восстановление копий журнала транзакций в базе данных Получателя. Сервер, который обслуживает базу данных Получателя, выступает в роли резервного сервера или сервера отчетов. Сервер отчетов предоставляет возможность обработки запросов пользователей на чтение данных. Одна база данных Источника может синхронизироваться с одной или с несколькими базами данных Получателей. Доставка журналов применима к тем базам данных, которые используют полную модель восстановления или модель восстановления с неполным протоколированием.

Давайте договоримся, что сервер, первоначально обслуживающий базу данных Источника, будем называть сервером источника (термин «основной сервер» был бы удобнее, но он используется в Зеркальном отображении, и мы его не станем использовать, чтобы не было путаницы), а сервер, первоначально обслуживающий базу данных Получателя, называть резервным сервером. После настройки, сервер источника обслуживает базу данных Источника, но может сменить роль и обслуживать базу данных Получателя. Изменение роли приводит к одновременному изменению настроек на всех участвующих в Доставке журналов серверах, поскольку сервер источника может быть только один.

В Доставке журналов предусмотрены средства мониторинга и оповещения обо всех штатных и нештатных состояниях участников процесса Доставки журналов. Начиная с SQL Server 2008, поддерживается сжатие резервных копий, и такие копии можно использовать для Доставки журналов. Копии журналов создаются и доставляются с помощью заданий автоматизации службы SQL Server Agent, всего таких заданий четыре: задание резервного копирования Источника, задание копирова-

ния файлов Получателю, задание восстановления копий на Получателе и задание рассылки предупреждений.

Формат хранения данных на диске для SQL Server x64 и x86 одинаков, т.е. в Доставке журналов могут участвовать сервера баз данных под управлением 32-х и 64-х разрядных операционных систем. Доставка журналов поддерживается следующими редакциями SQL Server 2005: Enterprise Edition, Standard Edition и Workgroup Edition. Серверы, задействованные в доставке журналов, должны иметь одинаковые параметры сортировки, а базы данных источника и получателей могут использовать только модель полного восстановления (Full) или модель восстановления с неполным протоколированием (Bulk Logged).

Доставка журналов может быть средством тиражирования данных, однако, в этой статье мы рассмотрим возможность использования Доставки журналов для первоначальной синхронизации баз данных в Зеркальном отображении и Репликации.

Из BOL мы знаем, что Доставку журналов допустимо совмещать с Зеркальным отображением. В таком случае база данных Источника должна являться основной базой данных Зеркального отображения. Причем, Зеркальное отображение может быть настроено в любом из трех режимов его работы. Базы данных Получателей не должны находиться на том же экземпляре сервера, что и база источника или на экземпляре Зеркального отображения базы Источника. Поскольку Зеркальное отображение позволяет делать резервные копии только базы данных основного сервера, использование в качестве базы данных Источника зеркальной копии основной базы невозможно. Настраивать Доставку журналов и Зеркальное отображение на основном сервере можно в любом порядке.

Допустимо совмещать Доставку журналов с Репликацией, и это справедливо для издаваемой базы данных, баз данных Подписок, и для базы данных Распространителя. Следует учитывать, что процесс репликации будет прерван в случае перехода на резервный сервер, поскольку агенты репликации не умеют правильно реагировать на изменение ролей серверов в Доставке журналов, и это приведет к тому, что транзакции перестанут тиражироваться подписчикам. После того, как роль базы данных Источника вернется в первоначальное состояние, репликация возобновляется, и все те транзакции, которые были скопированы Доставкой журналов с резервного сервера на сервер источник, реплицируются подписчикам. В случае безвозвратной потери базы данных Источника, достаточно переименовать Получателя, чтобы возобновить процесс репликации.

Первоначальная инициализация базы данных Получателя в процессе настройки Доставки журналов осуществляется путем восстановления полной резервной копии базы данных источника с параметрами NORECOVERY или STANDBY. Для организации доставки журналов необходимо создать сетевой ресурс, в котором будут создаваться резервные копии журналов транзакций и который будет доступен всем уча-

ствующим в Доставке журналов серверам. В нашем примере в качестве такого ресурса будет использоваться каталог сервера источника C:\MSSQL\LogShip\.

Существует возможность синхронизировать процесс доставки журналов с тиражированием данных в репликации транзакций. Такая синхронизация устанавливается с помощью системной хранимой процедуры sp_replicationdboption, которая позволяет для публикуемой базы данных и базы данных Распространителя установить опцию «sync with backup». Когда эта опция установлена для базы данных Распространителя, это гарантирует, что транзакции в журнале публикуемой базы данных не будут усечены до тех пор, пока не будет создана их резервная копия в базе данных Распространителя. Усечение журнала транзакций публикуемой базы данных откладывается до завершения резервного копирования усекаемых транзакций в базе данных Распространителя. Установка этой опции позволяет управлять точкой усечения базы данных Публикации. Новое значение вступает в силу после очередного запуска Агента чтения журнала или по истечении заданного параметром агента – MessageInterval интервала времени, если Агент чтения журнала работает в непрерывном режиме. За счет того, что в базе данных распространителя не будет транзакций, которых нет в резервной копии публикуемой базы данных, можно не боятся рассогласованности издаваемой базы данных и базы данных Распространителя в случае восстановления издаваемой базы данных из резервной копии. Однако, следует помнить, что такой способ синхронизации добавляет задержку в процесс тиражирования транзакций и не гарантирует отсутствия потерь данных, в случае повреждения журнала транзакций. Кроме того, для Доставки журналов опция важна только для издаваемой базы данных.

В рамках этой статьи мы не будем рассматривать совместную работу Доставки журналов и Репликации. Доставка журналов нами будет использоваться только для целей первоначальной синхронизации баз данных подписчиков и Зеркального отображения базы данных Издателя. Поэтому сценарии включения Доставки журналов будут использоваться в примерах с демонстрацией включения Зеркального отображения и Репликации.

Дополнительную информацию о Доставке журналов можно получить в следующих статьях BOL:

- Развертывание доставки журналов.
- Репликация и доставка журналов.
- Зеркальное отображение баз данных и доставка журналов.
- Резервное копирование и восстановление из копий реплицируемых баз данных.
- Стратегии резервного копирования и восстановления из копии репликации моментальных снимков и репликации транзакций.

Совместимость Зеркального отображения и Репликации

Не все базы данных в топологии Одноранговой репликации транзакций допускают Зеркальное отображение (зеркалирование). Например, это невозможно для базы данных Распространителя. В Репликации транзакций зеркалирование базы данных подписчика требует учета некоторых ограничений, которых нет для издаваемой базы или в одноранговой топологии. Возможность использования в репликации зеркалирования зависит от того, насколько будут способны задействованные агенты репликации отрабатывать состояние отказа основной базы данных и автоматически переключаться на зеркальную копию. Так, например, ни один из соответствующих агентов репликации не может правильно отреагировать на отказ базы данных Распространителя и переключиться на работу с зеркальной копией этой базы данных. Для обеспечения высокой доступности Распространителя его следует вынести на выделенный компьютер и отдать его под управление отказоустойчивого кластера. Однако, те агенты репликации, которые соединяются с публикуемой базой данных, умеют переключаться на зеркальную копию, и в случае отказа способны подключиться автоматически, обеспечив непрерывность процесса репликации.

С подписчиками все не так просто, как с издателем. К сожалению, ни один из работающих с базами на подписчике агентов репликации не предназначен для автоматического переключения в случае отказа. Самым естественным путем переключения потока данных репликации является удаление подписки и создание ее заново.

Дополнительные сведения можно получить в следующей статье BOL:

- Репликация и зеркальное отображение базы данных

Влияние зеркалирования на работу Агента чтения журнала

Зеркалирование публикуемой базы данных влияет на поведение Агента чтения журнала, его состояние становится зависимым от состояния зеркалирования. Агент чтения журнала зеркалируемой базы данных будет копировать из журнала вначале записи тех транзакций, которые были до этого скопированы и завершены в журнале регистрации транзакций зеркальной базы данных (процесс, с помощью которого пишутся записи в журнал транзакций зеркальной базы данных, в документации называют hardening – закрепление). Т.е. реплицироваться будут только записи с таким LSN, который больше LSN последней закрепленной в журнале транзакций зеркала.

Это позволяет выставить из топологии основной сервер (зеркало доступно, но существуют такие записи в журнале транзакций, которые еще

не были закреплены на зеркале) или изолировать его (когда зеркало недоступно). В обоих случаях, пока основной сервер работоспособен и его база данных доступна, любые изменения в его базе данных не будут реплицированы, пока соответствующие записи журнала транзакций не будут закреплены на зеркале.

Такое поведение добавляет задержки в поток репликации, и если произойдет отказ зеркалирования, будет гарантировано, что записи в журнале Подписчика не обгонят фиксацию в основной базе данных.

Ели Агент чтения журнала вынужден ждать закрепления записей в журнале транзакций зеркальной базы данных, в хронологии работы Агента чтения журнала будут появляться уведомления следующего вида: «Replicated transactions are waiting for next Log backup or for mirroring partner to catch up».

Изменение поведения Агента чтения журнала при установке флага трассировки 1448

В редких случаях, эффект задержек из-за зеркалирования для логики работы использующих репликацию приложений может оказаться не приемлем. Для решения этой проблемы добавлен новый флаг трассировки 1448, который предписывает репликации продолжаться даже в тех случаях, когда основная база данных выставлена или изолирована. Обычно, Агента чтения журнала ждет, пока не будут закреплены записи в журнале регистрации транзакций зеркальной базы данных, после чего он копирует их в базу данных Распространителя. Когда сервер запущен с флагом трассировки 1448, это ожидание исключается, и Агента чтения журнала может сразу копировать изменения, независимо от состояния зеркалирования.

Этот флаг трассировки можно применять в SQL Server 2008, а для SQL Server 2005 он появился в составе Cumulative update package 2 для SQL Server 2005 Service Pack 2. Подробности о флаге трассировки №1448 можно узнать в статье базы знаний Майкрософт №937041.

Обратите внимание на потенциальную опасность исключения задержек тиражирования в репликации. Для приложений отсутствие задержек может иметь решающее значение, и это может послужить причиной установки флага трассировки №1448 при использовании зеркалирования с репликацией. Надо понимать, что существует вероятность того, что использование этого флага приведет к проблемам в случае отказа Зеркального отображения. Эти проблемы будут рассмотрены ниже.

Влияние отказа зеркалирования на работу Агента чтения журнала

Когда происходит автоматическая или ручная отработка отказа Зеркального отображения, Агент чтения журнала должен автоматически соеди-

ниться с новым основным сервером и продолжить копировать транзакций (если параметр -PublisherFailoverPartner был установлен правильно, как это будет описано ниже). Есть два случая, когда этого может не произойти. Первый, это когда отказ не может быть отработан правильно, потому что зеркальный сервер по каким либо причинам не может стать основным сервером. Второй случай может произойти только тогда, когда включен флаг трассировки №1448, о котором только что упоминалось выше.

Когда Зеркальное отображение не работает в режиме высокой доступности или основной сервер был выставлен или изолирован, может оказаться, что на основном сервере есть завершенные транзакции, которых еще не были закреплены на зеркале. Если основная база данных станет недоступной, и произойдет передача ее роли зеркалу, может случиться потеря тех транзакций, которые были завершены на основном сервере, но еще не были закреплены на зеркале. Если к тому же включен флаг трассировки №1448, некоторые из завершенных транзакций основного сервера могут быть скопированы, но еще не закреплены на зеркале. Это приведет к тому, что состояние метаданных о тиражируемых транзакциях на Распространителе будет опережать реальное состояние транзакций на принявшем роль основной базы данных зеркале. Т.е. Агент чтения журнала уже передавал Распространителю некоторое количество транзакций. После обнаружения таких расхождений, Агент распространителя выдаст ошибку: «The process could not execute 'sp_repldone/sp_replcounters' on ' GLADCHENKO-TEST'». Кроме этого сообщения об ошибке, будут и детализирующие ситуацию сообщения, из которых можно узнать, какой номер виртуального журнала пытался считать Агент чтения журнала. Исправить положение можно запустив с помощью системной хранимой процедуры sp_replrestart принудительную синхронизацию метаданных Издателя и Распространителя. Однако, велика вероятность того, что из-за расхождений в метаданных часть не синхронизированных в результате отказа основного сервера транзакций уже может попасть в базы данных Подписчиков. Это приведет к тому, что Агент распространителя тоже прекратит свою работу, выдав сообщение об ошибке синхронизации данных. Поскольку в результате сбоя синхронизации Распространитель просто повторит изменения, передаваемые не прошедшими синхронизацию транзакциями, для преодоления ошибки работы Агента распространителя достаточно сменить его профиль на стандартный профиль с именем: «Continue on data consistency errors».

Как влияет нарушение зеркалирования на работу Агента чтения журнала

Как уже отмечалось, в случае отказа Зеркального отображения Агент чтения журнала станет работать с журналом транзакций зеркала основной базы данных. Возможность переключения на сменивший свою роль

новый основной сервер обеспечивается параметром запуска агента -PublisherFailoverPartner. Пока между обоими участниками Зеркального отображения действует партнерство, т.е. они продолжают работать по схеме основная база – зеркало основной базы, в случае отказа основной базы данных Агент чтения журнала подключится к принявшей роль основной, зеркальной базе данных.

Если основной сервер будет недоступен очень долго, лучше всего удалить зеркалирование, что позволит избежать проблем с ростом журнала транзакций зеркальной базы данных. Если отказ приведет к тому, что будет нарушено партнерство основной и зеркальной баз данных, Агент чтения журнала будет продолжать работать с зеркалом, пока ситуация с партнерством не восстановится или зеркальное отображение не будет удалено. Если потом агента перезапустить, он попробует соединяться с основной базой данных, которая больше не является издаваемой базой данных, что приведет к сбою репликации. В хронологии работы Агента чтения журнала появятся сообщения такого типа: «User-specified agent parameter values: -Publisher GLADCHENKO-TEST'». Также будут более детальные сообщения об отказе подключения. Для того чтобы решить подобную проблему, и заставить Агент чтения журнала продолжить свою работу, создайте на Распространителе псевдоним действующего сервера издателя.

Настройка периода хранения Распространителя

По умолчанию на Распространителе значение периода хранения транзакций установлено в ноль. Это означает, что транзакции очищаются сразу же, как только они будут доставлены всем Подписчикам. При использовании Зеркального отображения, в момент, когда из-за отказа Агент распространителя переключается на работу с зеркальной копией базы данных, нужно чтобы в базе данных распространителя еще оставались транзакции, которые уже доставлены всем Подписчикам. Такое случается, когда зеркалирование работает в высокопроизводительном режиме, т.е. передача транзакций зеркалу идет асинхронно и состояние зеркала может сильно отставать от состояния основной базы данных. Тогда, при необходимости добавления нового Подписчика, которого нежелательно инициализировать, у Агента распространителя должна существовать возможность догрузить новому Подписчику те транзакции, которые на момент добавления подписчика еще не успели закрепиться в зеркале. Другой такой случай может произойти при отказе сервера Издателя, когда данные в базе данных публикации больше не доступны, и это не позволяет корректно инициализировать новую базу данных подписчика. В таком случае, при отсутствии транзакций на Распространителе, может произойти потеря данных.

Чтобы избегать потери данных или длительного простоя в системе с зеркалированием базы данных подписчика, рекомендуется установить адекватный период задержки транзакций у распространителя.

В наших примерах мы будем использовать следующий вариант настройки параметра задержки распространения:

```
USE distribution
EXEC sp_changedistributiondb N'distribution', N'min_distretention', 3
EXEC sp_helpdistributiondb;
GO
```

Типы ручной синхронизации Подписчика

Репликация не поддерживает автоматический переход на зеркальное отображение подписанной на публикацию базы данных, т.е. хотя зеркалирование и возможно, Агент распространителя не сможет переключиться на зеркало после отказа основной базы данных Зеркального отображения. Работа Агента распространителя закончится ошибкой сразу, как только он поймет, что не может подключиться к базе данных, являющейся основной в партнерстве Зеркального отображения. Зеркалирование Подписчика можно настраивать как до, так и после подписки. Однако, если планируется инициализация Подписчика моментальным снимком Публикации, эффективнее будет настраивать Зеркальное отображение после инициализации базы данных Подписчика.

В случае отказа Зеркального отображения базы данных Подписчика, необходимо перенаправить Репликацию на зеркальный сервер, туда, где теперь располагается новая база данных Подписчика, на новом основном сервере. Хотя этот метод требует заново оформить подписку, полной ее инициализации не потребуется, и это благодаря новому типу синхронизации SQL Server 2008.

Все методы ручной инициализации Подписчика основаны на том, что база данных приводится в согласованное с Издателем состояние, и это становится отправной точкой тиражирования последовавших за этим состоянием транзакций Издателя. После создания Подписки с ручной синхронизацией, Агент распространителя должен иметь информацию о том, какие транзакции нужно тиражировать, не потеряв при этом новые данные или изменения в имеющихся данных. В качестве такой отправной точки используется тот номер LSN, который применим к соответствующему методу инициализации. Например, в методе с указанием параметра «replication support only» определение LSN происходит автоматически. Когда же инициализация осуществляется восстановлением из резервной копии, LSN берется из заголовка файла резервной копии. В случае инициализации по LSN, необходимый номер виртуального журнала задается пользователем. Этот номер нужно указывать аккуратно, т.к. относящиеся к нему транзакции должны присутствовать на этот момент в базе данных Распространителя. Вот почему так важна задержка транзакций у Распространителя, это очень помогает в случае отказа.

Примеры

Для демонстрации новых возможностей обеспечения доступности Репликации SQL Server 2008 нам потребуется сервер Распространитель, который в представленном ниже сценарии создания Распространителя называется GLADCHENKO-VHD:

```
– Сценарий создания Распространителя
use master
exec sp_adddistributor @distributor = N'GLADCHENKO-VHD'
, @password = N'' – безопасность репликации не тема этой статьи,
                  – оставляем уязвимой
GO
exec sp_adddistributiondb
  @database = N'distribution'
, @data_folder = N'C:\MSSQL\DATA'
, @log_folder = N'C:\LOG\DATA'
, @log_file_size = 2
, @min_distretention = 0
, @max_distretention = 72
, @history_retention = 48
, @security_mode = 1
GO

use [distribution]
if (not exists (select * from sysobjects where name = 'UIProperties'
     and type = 'U '))
     create table UIProperties(id int)
if (exists (select * from ::fn_listextendedproperty('SnapshotFolder'
     , 'user', 'dbo', 'table', 'UIProperties', null, null)))
     EXEC sp_updateextendedproperty
  N'SnapshotFolder'
     , N'C:\MSSQL\ReplData'
     , 'user'
     , dbo
     , 'table'
     , 'UIProperties'
else
     EXEC sp_addextendedproperty
  N'SnapshotFolder'
     , N'C:\MSSQL\ReplData'
     , 'user'
     , dbo
     , 'table'
     , 'UIProperties'
GO

exec sp_adddistpublisher
  @publisher = N'GLADCHENKO-VHD'
```

```
, @distribution_db = N'distribution'
, @security_mode = 1
, @working_directory = N'C:\MSSQL\ReplData'
, @trusted = N'false'
, @thirdparty_flag = 0
, @publisher_type = N'MSSQLSERVER'
GO
```

Зеркальное отображение издаваемой базы данных

Мы будем использовать сервер GLADCHENKO-TEST в качестве основного сервера в зеркале и сервера издателя в Репликации. Публиковаться будет база данных MIR, и она же будет иметь свое Зеркальное отображение. Публикация тоже будет называться MIR. Сервер, на котором будет размещаться зеркало, называется GLADCHENKO-VHD и этот же сервер является в нашей тестовой топологии репликации Распространителем. В качестве сервера Подписчика будет выступать экземпляр GLADCHENKO-A\SUB, на котором база данных подписки будет называться MIR. Хотя репликацию можно настроить с помощью SQL Server Management Studio, мы будем ее настраивать с помощью системных процедур Transact-SQL.

Для простоты демонстрации, все учетные записи, от имени которых будут запускаться службы, и имена входа, в контексте безопасности которых будут выполняться работы на серверах, будут сведены к одной, доменной учетной записи пользователя AG@troika.ru. Этот пользователь является локальным администратором на всех серверах, и включен в серверную роль sysadmins всех указанных выше серверов баз данных.

Шаг 1. Создание Публикации

Публикуемая база данных MIR состоит из четырех таблиц: PerformanceCounter, PerformanceSignature, PerformanceSignatureData и PerformanceSignatureHistory. На самом деле, эти таблицы тиражируются средствами репликации транзакций с сервера, на котором размещена база данных System Center Operations Manager 2007 SP1. Сделано это для того, чтобы эмулировать работу «живого» приложения, поскольку данные мониторинга инфраструктуры серверов поступают в эти таблицы постоянно и без длительных перерывов. Именно этот поток реплицируемой информации мы и будет реплицировать с сервера GLADCHENKO-TEST на сервер GLADCHENKO-A\SUB.

Для получения более подробной информации о настройке Распространителя, смотрите следующие темы в SQL Server 2008 Books Online.

- Как настроить публикацию и распространение (программирование репликации на языке Transact-SQL)
- How to: Configure Publishing and Distribution (SQL Server Management Studio)

Ниже представлен сценарий создания Публикации.

Вначале, нужно разрешить на Распространителе еще одного Издателя. Для этого на сервере GLADCHENKO-VHD выполните следующий сценарий:

```
exec sp_adddistpublisher
  @publisher = N'GLADCHENKO-TEST'
, @distribution_db = N'distribution'
, @security_mode = 1
, @working_directory = N'C:\MSSQL\ReplData'
, @trusted = N'false'
, @thirdparty_flag = 0
, @publisher_type = N'MSSQLSERVER'
GO
```

После этого можно приступить к созданию Публикации, для чего на сервере GLADCHENKO-TEST нужно выполнить следующий сценарий:

```
use master
-- Указываем Распространителя
exec sp_adddistributor @distributor = N'GLADCHENKO-VHD'
, @password = N'**********' – тут нужно указать заданный
                           – на Распространителе пароль
-- В ответ должны получить подтверждение:
-- "You have updated the Publisher property 'active' successfully"
GO

use [MIR]
exec sp_replicationdboption @dbname = N'MIR'
, @optname = N'publish'
, @value = N'true'
GO

-- Добавляет Публикацию
exec sp_addpublication
  @publication = N'MIR'
, @description = N'Публикация БД MIR в репликации
транзакций, на сервере GLADCHENKO-TEST'
, @sync_method = N'native'
, @retention = 0
, @allow_push = N'true'
, @allow_pull = N'true'
, @allow_anonymous = N'false'
, @enabled_for_internet = N'false'
, @snapshot_in_defaultfolder = N'true'
, @compress_snapshot = N'false'
, @ftp_port = 21
, @ftp_login = N'anonymous'
, @allow_subscription_copy = N'false'
, @add_to_active_directory = N'false'
```

```
, @repl_freq = N'continuous'
, @status = N'active'
, @independent_agent = N'true'
, @immediate_sync = N'true'
, @allow_sync_tran = N'false'
, @autogen_sync_procs = N'false'
, @allow_queued_tran = N'false'
, @allow_dts = N'false'
, @replicate_ddl = 1
, @allow_initialize_from_backup = N'true'
, @enabled_for_p2p = N'false'
, @enabled_for_het_sub = N'false'
, @p2p_conflictdetection = N'false'
, @p2p_originator_id = 1
-- В ответ должны получить сообщение:
-- Job 'GLADCHENKO-TEST-MIR-1' started successfully.
-- "Warning: The logreader agent job has been implicitly created
-- and will run under the SQL Server Agent Service Account"
GO

-- Создаем инициализирующий моментальный снимок
exec sp_addpublication_snapshot @publication = N'MIR'
, @frequency_type = 1
, @frequency_interval = 1
, @frequency_relative_interval = 1
, @frequency_recurrence_factor = 0
, @frequency_subday = 8
, @frequency_subday_interval = 1
, @active_start_time_of_day = 0
, @active_end_time_of_day = 235959
, @active_start_date = 0
, @active_end_date = 0
, @job_login = null
, @job_password = null
, @publisher_security_mode = 1
GO

-- Добавляем наши таблицы в виде статей публикации
exec sp_addarticle @publication = N'MIR'
, @article = N'PerformanceCounter'
, @source_owner = N'dbo'
, @source_object = N'PerformanceCounter'
, @type = N'logbased'
, @description = null
, @creation_script = null
, @pre_creation_cmd = N'drop'
, @schema_option = 0x000000000803509F
, @identityrangemanagementoption = N'manual'
, @destination_table = N'PerformanceCounter'
, @destination_owner = N'dbo'
```

```
, @vertical_partition = N'false'
, @ins_cmd = N'CALL sp_MSins_dboPerformanceCounter'
, @del_cmd = N'CALL sp_MSdel_dboPerformanceCounter'
, @upd_cmd = N'SCALL sp_MSupd_dboPerformanceCounter'
GO
exec sp_addarticle @publication = N'MIR'
, @article = N'PerformanceSignature'
, @source_owner = N'dbo'
, @source_object = N'PerformanceSignature'
, @type = N'logbased'
, @description = null
, @creation_script = null
, @pre_creation_cmd = N'drop'
, @schema_option = 0x000000000803509F
, @identityrangemanagementoption = N'manual'
, @destination_table = N'PerformanceSignature'
, @destination_owner = N'dbo'
, @vertical_partition = N'false'
, @ins_cmd = N'CALL sp_MSins_dboPerformanceSignature'
, @del_cmd = N'CALL sp_MSdel_dboPerformanceSignature'
, @upd_cmd = N'SCALL sp_MSupd_dboPerformanceSignature'
GO
exec sp_addarticle @publication = N'MIR'
, @article = N'PerformanceSignatureData'
, @source_owner = N'dbo'
, @source_object = N'PerformanceSignatureData'
, @type = N'logbased'
, @description = null
, @creation_script = null
, @pre_creation_cmd = N'drop'
, @schema_option = 0x000000000803509F
, @identityrangemanagementoption = N'manual'
, @destination_table = N'PerformanceSignatureData'
, @destination_owner = N'dbo'
, @vertical_partition = N'false'
, @ins_cmd = N'CALL sp_MSins_dboPerformanceSignatureData'
, @del_cmd = N'CALL sp_MSdel_dboPerformanceSignatureData'
, @upd_cmd = N'SCALL sp_MSupd_dboPerformanceSignatureData'
GO
exec sp_addarticle @publication = N'MIR'
, @article = N'PerformanceSignatureHistory'
, @source_owner = N'dbo'
, @source_object = N'PerformanceSignatureHistory'
, @type = N'logbased'
, @description = null
, @creation_script = null
, @pre_creation_cmd = N'drop'
, @schema_option = 0x000000000803509F
, @identityrangemanagementoption = N'manual'
, @destination_table = N'PerformanceSignatureHistory'
```

```
, @destination_owner = N'dbo'
, @vertical_partition = N'false'
, @ins_cmd = N'CALL sp_MSins_dboPerformanceSignatureHistory'
, @del_cmd = N'CALL sp_MSdel_dboPerformanceSignatureHistory'
, @upd_cmd = N'SCALL sp_MSupd_dboPerformanceSignatureHistory'
GO

-- настраиваем и запускаем Агента чтения журнала
exec [MIR].sys.sp_addlogreader_agent
  @job_login = null
, @job_password = null
, @publisher_security_mode = 1
GO
exec [MIR].sys.sp_addqreader_agent
  @job_login = null
, @job_password = null
, @frompublisher = 1
GO

-- Формируем лист доступа к публикации (PAL)
-- Сюда нужно даобавить все задействованные агентами логины
exec sp_grant_publication_access
  @publication = N'MIR'
, @login = N'sa'
GO
exec sp_grant_publication_access
  @publication = N'MIR'
, @login = N'NT AUTHORITY\SYSTEM'
GO
```

Шаг 2. Настройка Доставки журналов для инициализации Подписчика

На этом шаге мы должны подготовить базу данных на сервере Подписчика, для того, чтобы она могла быть инициализирована из резервной копии. Поскольку поток транзакций в базе данных нашего Издателя не прерывается 24 часа в сутки, 7 дней в неделю и 365 дней в году, нам будет удобно задействовать для этого механизм Доставки журналов.

Для включения Доставки журналов нужно инициализировать базу данных Получателя путем восстановления полной резервной копии базы данных Источника. В нашем случае, резервным сервером, а заодно и сервером Подписчика будет компьютер GLADCHENKO-A\SUB.

Делаем полную копию издаваемой базы данных, для чего, на сервере GLADCHENKO-TEST выполняем следующий сценарий:

```
BACKUP DATABASE [MIR]
TO  DISK = N'C:\MSSQL\BACKUP\MIR.bak'
WITH NOFORMAT, INIT
```

```
, NAME = N'Полная копия БД MIR'
, SKIP, NOREWIND, NOUNLOAD, STATS = 10
GO
```

Дожидаемся получения подтверждения успешности этой операции, в окне результатов появится приблизительно такой текст:

```
10 percent processed.
20 percent processed.
30 percent processed.
40 percent processed.
50 percent processed.
60 percent processed.
70 percent processed.
80 percent processed.
90 percent processed.
Processed 18104 pages for database 'MIR', file 'MIR' on file 1.
100 percent processed.
Processed 3 pages for database 'MIR', file 'MIR_log' on file 1.
BACKUP DATABASE successfully processed 18107 pages in 4.394 seconds (32.193 MB/sec).
```

После того, как копия успешно создана, нужно ее скопировать на сервер Источника. В нашем случае, нужно взять файл с компьютера GLADCHENKO-TEST из папки C:\MSSQL\BACKUP\MIR.bak и скопировать его на компьютер GLADCHENKO-A, в папку: C:\MSSQL\BACKUP\MIR.bak.

После того, как файл резервной копии окажется на сервере Получателя, нужно восстановить базу данных Получателя, что мы и сделаем на экземпляре GLADCHENKO-A\SUB, используя следующий сценарий:

```
RESTORE DATABASE [MIR]
FROM  DISK = N'D:\MSSQL\BACKUP\MIR.bak'
WITH  FILE = 1
,  MOVE N'MIR' TO N'C:\MSSQL\DATA\MIR.mdf'
,  MOVE N'MIR_log' TO N'D:\MSSQL\LOG\MIR_1.ldf'
,  NOUNLOAD,  REPLACE,  STATS = 10, NORECOVERY
GO
```

После успешного восстановления базы данных, приступаем к настройке Доставки журналов между публикуемой базой данных на Издателе, и только что восстановленной базой на Подписчике. Подробное описание настройки Доставки журналов можно найти в статье электронной документации: «Как включить доставку журналов (Transact-SQL)».

Создадим папку для обмена резервными копиями журналов транзакций: C:\MSSQL\LogShip. На эту папку у учетных записей служб должен быть полный доступ.

Далее, настраивает базу данных Источника вместе с заданием резервного копирования, а также записями локального и удаленного мониторов:

```
DECLARE @LS_BackupJobId AS uniqueidentifier
DECLARE @LS_PrimaryId AS uniqueidentifier
```

```
EXEC master.dbo.sp_add_log_shipping_primary_database
 @database = N'MIR'
,@backup_directory = N'C:\MSSQL\LogShip'
,@backup_share = N'\\GLADCHENKO-TEST\lsbackup'
,@backup_job_name = N'LSBackup_MIR'
,@backup_retention_period = 1440
,@backup_compression = 0
,@monitor_server = N'GLADCHENKO-A'
,@monitor_server_security_mode = 1
,@backup_threshold = 60
,@threshold_alert_enabled = 1
,@history_retention_period = 1440
,@backup_job_id = @LS_BackupJobId OUTPUT
,@primary_id = @LS_PrimaryId OUTPUT
,@overwrite = 1
GO
```

Значения идентификаторов получились соответственно следующие:

```
@LS_BackupJobId = DDCD7E77-F045-4649-A9F1-CF8FE843C711
@LS_PrimaryId = 3346F94D-3DB6-4A56-BE99-DF2E60E0A25F
```

Если процедура отработала без ошибок, должно появиться новое задание, предусматривающее один шаг, в котором выполняется следующая команда:

```
c:\Program Files\Microsoft SQL Server\100\Tools\Binn\sqllogship.exe» -Backup
3346F94D-3DB6-4A56-BE99-DF2E60E0A25F -server GLADCHENKO-TEST'
```

Создаем расписание для задания резервного копирования журналов транзакций базы данных MIR:

```
DECLARE @schedule_id int
EXEC msdb.dbo.sp_add_jobschedule
 @job_id=N'DDCD7E77-F045-4649-A9F1-CF8FE843C711'
,@name=N'1'
,@enabled=1
,@freq_type=4
,@freq_interval=1
,@freq_subday_type=4
,@freq_subday_interval=10
,@freq_relative_interval=0
,@freq_recurrence_factor=1
,@active_start_date=20081202
,@active_end_date=99991231
,@active_start_time=0
,@active_end_time=235959
,@schedule_id = @schedule_id OUTPUT
SELECT @schedule_id
GO
```

Это расписание предусматривает выполнение задания каждые 10 минут в течение суток.

Мы не планируем создавать полное решение Доставки журналов, наша задача только облегчить первоначальную инициализацию Подписчика. Однако, давайте полностью следовать упомянутой выше инструкции BOL по включению Доставки журналов.

Далее, нам нужно на сервере Источника задать базу данных Получателя:

```
EXEC master.dbo.sp_add_log_shipping_primary_secondary
 @primary_database = N'MIR'
,@secondary_server = N'GLADCHENKO-A\SUB'
,@secondary_database = N'MIR'
,@overwrite = 1
GO
```

После этого, на сервере Получателя (Подписчик, в нашем примере это компьютер GLADCHENKO-A\SUB) нужно запустить процедуру, которая создаст все необходимые задания по применению резервных копий на базе данных Получателя.

```
DECLARE @LS_Secondary__CopyJobId      AS uniqueidentifier
DECLARE @LS_Secondary__RestoreJobId   AS uniqueidentifier
DECLARE @LS_Secondary__SecondaryId    AS uniqueidentifier

EXEC master.dbo.sp_add_log_shipping_secondary_primary
 @primary_server = N'GLADCHENKO-TEST'
,@primary_database = N'MIR'
,@backup_source_directory = N'\\GLADCHENKO-TEST\lsbackup'
,@backup_destination_directory = N'\\GLADCHENKO-A\D$\MSSQL\LogShip'
,@copy_job_name = N'LSCopy_GLADCHENKO-TEST_MIR'
,@restore_job_name = N'LSRestore_GLADCHENKO-TEST_MIR'
,@file_retention_period = 4320
,@monitor_server = N'GLADCHENKO-A'
,@monitor_server_security_mode = 1
,@overwrite = 1
,@copy_job_id = @LS_Secondary__CopyJobId OUTPUT
,@restore_job_id = @LS_Secondary__RestoreJobId OUTPUT
,@secondary_id = @LS_Secondary__SecondaryId OUTPUT
GO
```

Получаем следующий набор идентификаторов:

```
@LS_Secondary__CopyJobId  = 338E7DF6-F266-40CD-902E-86F424E12E6C
@LS_Secondary__RestoreJobId = 6C0FE890-5FC5-4BA7-BD89-EC13214514E8
@LS_Secondary__SecondaryId = DC7800CE-767C-4F2B-9A42-70DC5F807184
```

После этого, среди заданий экземпляра GLADCHENKO-A\SUB появятся два новых задания, для которых определены шаги, но еще не настроены расписания их работы. Первое задание с именем:

LSCopy_GLADCHENKO-TEST_MIR. У этого задания только один шаг и он выполняет следующий сценарий:

```
C:\Program Files\Microsoft SQL Server\100\Tools\Binn\sqllogship.exe -Copy DC7800CE-
767C-4F2B-9A42-70DC5F807184 -server GLADCHENKO-A\SUB
```

Эта инструкция копирует файлы резервных копий журналов транзакций с сервера Источника на сервер Получателя.

Добавить типовое расписание работы этого задания можно с помощью следующего сценария:

```
EXEC msdb.dbo.sp_add_schedule
 @schedule_name =N'DefaultCopyJobSchedule'
,@enabled = 1
,@freq_type = 4
,@freq_interval = 1
,@freq_subday_type = 4
,@freq_subday_interval = 15
,@freq_recurrence_factor = 0
,@active_start_date = 20081203
,@active_end_date = 99991231
,@active_start_time = 0
,@active_end_time = 235900
,@schedule_uid = @LS_SecondaryCopyJobScheduleUID OUTPUT
,@schedule_id = @LS_SecondaryCopyJobScheduleID OUTPUT

EXEC msdb.dbo.sp_attach_schedule
 @job_id = '338E7DF6-F266-40CD-902E-86F424E12E6C'
,@schedule_id = @LS_SecondaryCopyJobScheduleID
GO
```

Второе задание называется LSRestore_GLADCHENKO-TEST_MIR и предназначено для восстановления скопированных с Источника резервных копий на базе данных Получателя. Единственный шаг этого задания выполняет следующую инструкцию:

```
C:\Program Files\Microsoft SQL Server\100\Tools\Binn\sqllogship.exe -Restore
DC7800CE-767C-4F2B-9A42-70DC5F807184 -server GLADCHENKO-A\SUB
```

Для добавления этому заданию расписания запуска необходимо на сервере GLADCHENKO-A\SUB выполнить следующий сценарий:

```
EXEC msdb.dbo.sp_add_schedule
 @schedule_name =N'DefaultRestoreJobSchedule'
,@enabled = 1
,@freq_type = 4
,@freq_interval = 1
,@freq_subday_type = 4
,@freq_subday_interval = 15
,@freq_recurrence_factor = 0
,@active_start_date = 20081203
,@active_end_date = 99991231
```

```
,@active_start_time = 0
,@active_end_time = 235900
,@schedule_uid = @LS_SecondaryRestoreJobScheduleUID OUTPUT
,@schedule_id = @LS_SecondaryRestoreJobScheduleID OUTPUT

EXEC msdb.dbo.sp_attach_schedule
 @job_id = '6C0FE890-5FC5-4BA7-BD89-EC13214514E8'
,@schedule_id = @LS_SecondaryRestoreJobScheduleID
GO
```

И теперь осталось только добавить в Доставку журналов базу данных Получателя, что можно сделать, выполнив следующий сценарий на сервере GLADCHENKO-A\SUB:

```
EXEC master.dbo.sp_add_log_shipping_secondary_database
 @secondary_database = N'MIR'
,@primary_server = N'GLADCHENKO-TEST'
,@primary_database = N'MIR'
,@restore_delay = 0
,@restore_mode = 0
,@disconnect_users          = 0
,@restore_threshold = 45
,@threshold_alert_enabled = 1
,@history_retention_period = 1440
,@overwrite = 1
GO
```

Следующим действием в настройке Доставки журналов будет подготовка сервера мониторинга, в качестве которого будет выступать экземпляр GLADCHENKO-A. На этом экземпляре нужно выполнить следующий сценарий:

```
EXEC msdb.dbo.sp_processlogshippingmonitorprimary
 @mode = 1
,@primary_id = N'3346F94D-3DB6-4A56-BE99-DF2E60E0A25F'
,@primary_server = N' GLADCHENKO-TEST'
,@monitor_server = N'GLADCHENKO-A'
,@monitor_server_security_mode = 1
,@primary_database = N'MIR'
,@backup_threshold = 60
,@threshold_alert = 14420
,@threshold_alert_enabled = 1
,@history_retention_period = 1440
GO
```

После этого, нужно включить те задания обслуживания Доставки журналов, которые были отмечены как выключенные. Это должно привести к тому, что Доставка журналов начнет свою работу и будет поддерживать актуальность базы данных MIR на сервере GLADCHENKO-A\SUB в автоматическом режиме.

Теперь, когда придет время создавать Подписку, достаточно будет ненадолго отключить клиентов от базы данных Издателя, дождаться или запустить вручную поочередно все задания Доставки журналов на Источнике и потом на Получателе, отключить все эти задания, и подготовить базу данных Получателя к настройке Подписки, выполнив этот сценарий:

```
RESTORE DATABASE MIR WITH RECOVERY
GO
```

Шаг 3. Настройка Подписки

После успешной инициализации базы данных Подписчика резервной копией, которую мы поддерживали в актуальном состоянии средствами Доставки журналов, можно приступать к подписке на Публикацию. Для этого, на сервере GLADCHENKO-TEST нужно с помощью представленного ниже сценария объявить сервер и базу данных Подписчика:

```
use [MIR]
exec sp_addsubscription @publication = N'MIR'
, @subscriber = N'GLADCHENKO-A\SUB'
, @destination_db = N'MIR'
, @sync_type = N'initialize with backup'
, @backupdevicetype = N'disk'
, @backupdevicename = 'LogShip\MIR_20081203195807.trn'
, @fileidhint = 1
, @subscription_type = N'pull'
, @update_mode = N'read only'
GO
```

Все последующие действия по оформлению Подписки нужно выполнять на сервере GLADCHENKO-A\SUB. Следующий сценарий создает Подписку по запросу:

```
use [MIR]
exec sp_addpullsubscription @publisher = N'GLADCHENKO-TEST'
, @publication = N'MIR'
, @publisher_db = N'MIR'
, @independent_agent = N'True'
, @subscription_type = N'pull'
, @description = N''
, @update_mode = N'read only'
, @immediate_sync = 1
```

И наконец, следующий сценарий нужно выполнить для создания задания запуска Агента распространителя, который будет доставлять изменения непосредственно Подписчику:

```
exec sp_addpullsubscription_agent @publisher = N'GLADCHENKO-TEST'
, @publisher_db = N'MIR'
, @publication = N'MIR'
```

```
, @distributor = N'GLADCHENKO-VHD'
, @distributor_security_mode = 1
, @distributor_login = N''
, @distributor_password = null
, @enabled_for_syncmgr = N'False'
, @frequency_type = 64
, @frequency_interval = 0
, @frequency_relative_interval = 0
, @frequency_recurrence_factor = 0
, @frequency_subday = 0
, @frequency_subday_interval = 0
, @active_start_time_of_day = 0
, @active_end_time_of_day = 235959
, @active_start_date = 20081203
, @active_end_date = 99991231
, @alt_snapshot_folder = N''
, @working_directory = N''
, @use_ftp = N'False'
, @job_login = null
, @job_password = null
, @publication_type = 0
GO
```

Когда все необходимые действия по оформлению подписки завершены, на сервер GLADCHENKO-A\SUB можно полностью удалить Доставку журналов и позаботиться об удалении созданных там заданий Доставки журналов.

Шаг 4. Зеркалирование публикуемой базы данных

В репликации транзакций существует очень опасная точка отказа, это Издатель, об обеспечении высокой доступности которого раньше было трудно заботиться. Теперь, для повышения доступности Издателя, можно создать Зеркальное отображение его базы данных и обеспечить автоматическое переключение на зеркальную копию в случае отказа. Для примера, мы попробуем настроить Зеркальное отображение базы данных Публикации на сервере Издателя и имитируем отказ с переключением Агента чтения журнала на зеркальную копию публикуемой базы данных.

В электронной документации BOL есть несколько статей, которые подробно рассматривают вопросы, связанные с настройкой Зеркального отображения баз данных. Вот список названий некоторых из них:

- Настройка зеркального отображения базы данных.
- Как настроить сеанс зеркального отображения базы данных (среда SQL Server Management Studio).
- Подготовка зеркальной базы данных к зеркальному отображению.
- Пример. Настройка зеркального отображения базы данных с помощью проверки подлинности Windows (язык Transact-SQL).

- Зеркальное отображение базы данных ALTER DATABASE (Transact-SQL).

В последней из перечисленных выше статей говориться, что предварительно на зеркальном сервере должна быть восстановлена резервная копия основной базы данных, и все последующие резервные копии журнала транзакций, восстановленные с использованием предложения WITH NORECOVERY. Как уже отмечалось выше, лучше всего подготовку синхронной копии зеркалируемой базы данных позволяет сделать Доставка журналов. Поскольку в настоящей статье мы уже уделили много внимания настройке Доставки журналов, повторение этой процедуры для инициализации копии основной базы данных мы опустим, будем считать, что все эти процедуры уже выполнены. Копия базы данных на момент начала настройки Зеркального отображения должна находиться в актуальном по отношению к основной базе данных состоянии.

В нашем примере, основной экземпляр и зеркало настраиваются так, чтобы использовать одного и того же Распространителя. В конфигурации Агента чтения журнала мы изменим настройки, чтобы получить нужное нам поведение агента в случае отказа сервера Издателя. Для этого на сервере Распространителя GLADCHENKO-VHD нужно установить значение для параметра запуска Агент чтения журнала – PublisherFailoverPartner, например так:

```
exec sp_add_agent_parameter
  @profile_id = 1 -- ID Агента моментальных снимков
, @parameter_name = N'-PublisherFailoverPartner'
, @parameter_value = N'GLADCHENKO-VHD' -- сервер зеркала
GO

exec sp_add_agent_parameter
  @profile_id = 2 -- ID Агента чтения журнала
, @parameter_name = N'-PublisherFailoverPartner'
, @parameter_value = N'GLADCHENKO-VHD' -- сервер зеркала
GO
```

Для принятия изменений профиля, нужно перезапустить задание Агента чтения журнала.

Зеркалирование публикуемой базы данных начнем с создания конечных точек зеркалирования. В первую очередь создадим конечную точку на Издателе, это у нас компьютер GLADCHENKO-TEST:

```
CREATE ENDPOINT [MirroringEndpoint]
    AUTHORIZATION [AG@troika.ru]
    STATE=STARTED
    AS TCP (LISTENER_PORT = 5022, LISTENER_IP = ALL)
    FOR DATA_MIRRORING
    (    ROLE = PARTNER
   , AUTHENTICATION = WINDOWS KERBEROS
```

```
    , ENCRYPTION = REQUIRED ALGORITHM RC4
    )
GO
```

На сервере Зеркального отображения GLADCHENKO-VHD (это у нас Распространитель) тоже нужна конечная точка:

```
CREATE ENDPOINT [MirroringEndpoint]
    AUTHORIZATION [AG@troika.ru]
    STATE=STARTED
    AS TCP (LISTENER_PORT = 5022, LISTENER_IP = ALL)
    FOR DATA_MIRRORING
    (    ROLE = ALL
    , AUTHENTICATION = WINDOWS KERBEROS
        , ENCRYPTION = REQUIRED ALGORITHM RC4
    )
GO
```

Проверить, что конечная точка была успешно создана и запущена, можно с помощью следующей инструкции:

```
SELECT role_desc, state_desc FROM sys.database_mirroring_endpoints
```

Одного создания конечных точек недостаточно, необходимо еще выдать права на работу с ними тем именам входа, которые будут к ним подключаться. Пример соответствующей инструкции ниже:

```
GRANT CONNECT ON ENDPOINT:: MirroringEndpoint TO AG@troika.ru
GO
```

После раздачи прав, нужно объявить участников Зеркального отображения. На сервере Зеркальной копии GLADCHENKO-VHD выполним следующую команду:

```
ALTER DATABASE [MIR] SET PARTNER = 'TCP://GLADCHENKO-TEST.troika.ru:5022'
```

После этого, с основного сервера GLADCHENKO-TEST запустим симметричную команду:

```
ALTER DATABASE [MIR] SET PARTNER = 'TCP://GLADCHENKO-VHD.troika.ru:5022'
```

После этого должна произойти синхронизация основной и зеркальной баз данных. Зеркальное отображение должно быть запущено. Если этого не произошло, попробуйте внимательно изучить сообщение об ошибке. Чаще всего, причина в несогласованности баз, т.е. недостает последних изменений, которые могли быть не перенесены Доставкой журналов, или нечто подобное. Если причина не в этом, попробуйте отменить роль Зеркала и повторить настройку сначала. Сценарий отмены роли Зеркала представлен ниже:

```
ALTER DATABASE [MIR] SET PARTNER OFF
```

Современные серверные операционные системы оснащаются встроенными Брандмауэрами. Такие сетевые экраны тоже способны помешать подключению к порту конечной точки. Проверить возможность подключения с основного сервера к серверу зеркальной копии можно с помощью консольной команды операционной системы:

```
telnet GLADCHENKO-VHD 5022
```

Шаг 5. Переключение Публикации на зеркальную копию

Ну а теперь мы перейдем к самому интересному, ради чего были все эти подготовительные действия. Итак, мы имеем репликацию транзакций между двумя серверами, и Распространителя на третьем сервере. Кроме того, у нас настроено Зеркальное отображение базы данных Издателя на сервер Распространителя. Перед нами стоит задача – возобновить репликацию, которая была прервана в результате отказа Издателя. Доступность Издателя определяется Зеркальным отображением базы данных, на основе объектов которой создана наша Публикация. Отказ этой базы данных расценивается нами, как отказ Зеркального отображения и поводом для ручной или автоматической смены роли, когда зеркальная копия становится основной базой данных.

Далее, имитируем отказ основного сервера, который у нас является Издателем:

```
ALTER DATABASE MIR SET PARTNER FAILOVER
```

В ответ получаем сообщение: «Nonqualified transactions are being rolled back. Estimated rollback completion: 100%».

После этого основная база данных Зеркального отображения меняется ролями со своим зеркалом. Причем, на Распространителе появляется публикация и видно, что у нее уже существует Подписка (которую мы создавали на сервере GLADCHENKO-TEST). Изменения, которые пользователи будут делать в базе данных MIR после ее переезда на сервер GLADCHENKO-VHD, будут вначале закрепляться в новом зеркале, в базе данных на сервере GLADCHENKO-TEST. Потом они будут попадать в базу данных Распространителя, считанные Агентом чтения журнала, который умеет обращаться ко второму партнеру Зеркального отображения в случае отказа основного сервера или основной базы данных.

Если на сервере GLADCHENKO-TEST запустим следующую команду:

```
SELECT PUBLISHINGSERVERNAME()
```

Эта команда возвратит имя исходного издателя для опубликованной базы данных, участвующей в Зеркальном отображении. Эта функция показывает первоначального издателя опубликованной базы данных, и поэтому у нас она вернет, и всегда будет возвращать: «GLADCHENKO-TEST».

Если после этого выполнить на сервере GLADCHENKO-VHD ту же саму команду: «ALTER DATABASE MIR SET PARTNER FAILOVER», то все вернется в изначальное состояние. Публикация снова будет располагаться на сервере GLADCHENKO-TEST.

В отличие от агентов репликации, Монитор репликации не очень приспособлен для отслеживания работы с перемещенной на зеркальный сервер базой данных. Хотя его интерфейс будет показывать перемещение Публикации, информация о сеансах репликации может вводить администратора репликации в заблуждение. Трассировочные маркеры тоже не всегда помогают контролировать процесс репликации. В таком режиме работы репликации лучше всего контролировать сами данные, сверяя последние их изменения на издателе и подписчиках.

Существует опасность, что отказ основной базы данных приведет к возникновению проблем репликации транзакций. Это особенно актуально, когда не соблюдается порядок первоначальной фиксации транзакции в зеркале базе данных. Если случиться так, что несколько некорректных транзакций будут препятствовать продолжению репликации данных, можно принудительно проигнорировать эти транзакции, что позволяет сделать системная процедура sp_setsubscriptionxactseqno.

Выводы

Зеркальное отображение можно использовать для повышения доступности Издателя в топологиях Репликации транзакций и Репликации слияния. Если важно, чтобы клиенты продолжали получать возможность изменения данных в издаваемой базе данных и автоматически переключались на ее зеркальную копию в случае отказа, применение зеркалирования является оправданным.

В упомянутом в самом начале настоящей статьи техническом документе Майкрософт «SQL Server Replication: Providing High Availability using Database Mirroring» предлагается вариант решения для зеркалирования баз данных Подписчиков. Там же вы найдете аргументы для выбора такого решения.

Для повышения доступности базы данных Распространителя существует пока только одно решение – отказоустойчивая кластеризация.

Настройка транзакционной репликации в SQL Server 2005

По материалам статьи Байя Павлиашвили (Baya Pavliashvili) «Setting Up Transactional Replication with SQL Server 2005»

Перевод предоставлен Дмитрием Артемовым

В Microsoft SQL Server можно реплицировать данные и изменения схемы с одного сервера баз данных на другой. Байя Павлиашвили расскажет об использовании мастеров SQL Server 2005 и сценариев для настройки транзакционной репликации, выделив особо некоторые из самых полезных новых функций в последней версии.

Репликация – это передовая функция Microsoft SQL Server, которая позволяет переносить данные и изменения схемы с одного сервера баз данных на другой. В этой статье я продемонстрирую, как настроить транзакционную репликацию в простой среде на основе SQL Server 2005. В следующих статьях этой серии мы обсудим обслуживание и решение проблем при транзакционной репликации, а также репликацию модулей кода.

Примечание. Примеры в этой статье приведены в расчете на SQL Server 2005 Service Pack 1. Хотя я не собиралась проводить здесь сравнение функциональности версий 2000 и 2005, я подчеркну те улучшения в новой версии, на которые, по-моему, стоит обратить внимание. Прочитав статью, вы научитесь настраивать транзакционную репликацию с помощью мастеров и сценариев. В большинстве случаев для изначальной настройки публикации и подписчиков вы будете пользоваться мастерами, однако если вам потребуется одинаковым образом настроить публикацию в нескольких средах, вы оцените возможность применения сценариев по достоинству.

Настройка распределительного сервера (Distributor)

В SQL Server 2005 представлены многочисленные улучшения в области репликации, не самым последним из которых являются более короткие мастера. Следовать указаниям мастера вовсе не сложно, но чем меньше в нем этапов, тем быстрее выполняется настройка репликации. Как правило, связанные с репликацией мастера в SQL Server 2005 примерно на 50% короче, чем в SQL Server 2000.

Первый шаг при конфигурировании репликации – назначение сервера для хранения и доставки реплицированных транзакций – распределительного сервера или дистрибьютора. Один и тот же сервер может одновременно являться публикующим сервером (издателем-Publisher), распределительным сервером и сервером-подписчиком. Однако в реальном сценарии выполнять функции издателя и подписчика будут, скорее всего, два разных сервера. Использование отдельного сервера в качестве дистрибьютора поможет снизить нагрузку на издателя.

Для запуска Configure Distribution Wizard подключитесь к нужному экземпляру SQL Server с помощью SQL Server Management Studio (SSMS), найдите папку replication, щелкните по ней правой кнопкой мыши и выберите в контекстном меню пункт Configure Distribution. Мастера настройки репликации больше не являются модальными, то есть вы можете продолжать работу с SSMS при активном мастере. Первый экран мастера просто информирует вас о задачах, которые вы можете выполнить с его помощью. Если в дальнейшем вы не хотите снова видеть этот экран, поставьте соответствующий флажок.

На следующем экране вам нужно выбрать, использовать в качестве дистрибьютора локальный сервер или какой-либо другой (см. рис. 1).

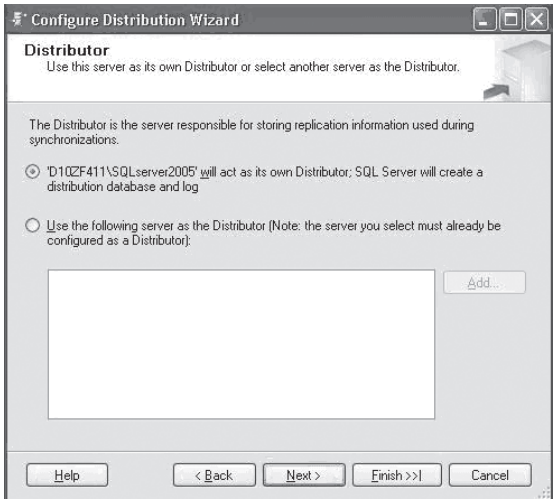


Рис. 1

Если вы хотите назначить дистрибьютором удаленный сервер, то сначала нужно запустить на том сервере Configure Distribution Wizard. В данном примере я буду использовать один и тот же экземпляр SQL Server в качестве публикующего и распределительного серверов. Следующий экран позволяет указать папку для мгновенных снимков, где будут храниться данные и схема опубликованной базы данных (см. рис. 2). По умолчанию эта папка называется ReplData и создается в каталоге установки текущего экземпляра SQL Server.

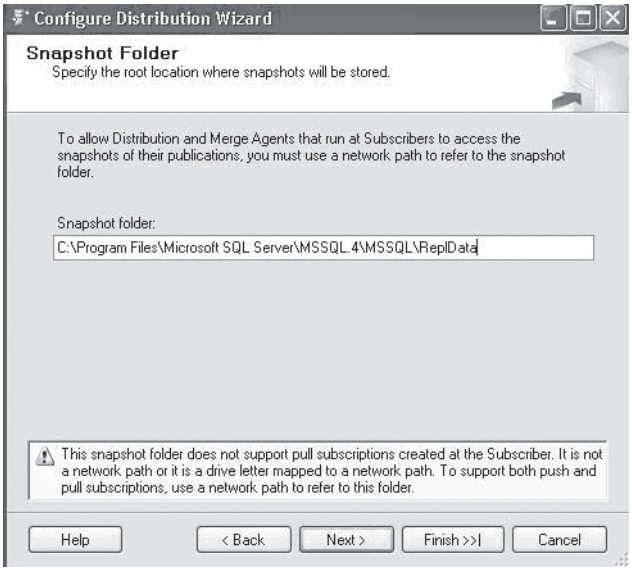


Рис. 2

Обратите внимание на диалоговое окно с предупреждением о том, что текущий каталог не поддерживает запросную (pull) подписку. Для использования запросной подписки вам потребуется сетевая папка для хранения снимков. Поскольку в моем примере издатель и подписчик будут расположены на одном компьютере, я могут спокойно проигнорировать это сообщение и просто щелкнуть Next.

На следующем экране нужно указать имя распределительной базы данных и местоположение для ее данных и файлов журнала транзакций. По умолчанию распределительная база данных называется distribution, но у вас могут быть причины изменить это имя. Например, если у вас сотни публикаций, то вы, возможно, захотите создать несколько распределительных баз данных и дать каждой из них осмысленное имя. Для размещения базы данных и файлов журнала мастер будет использовать стандартное местоположение. Изменить его можно на вкладке Database Settings диалогового окна Server Properties в SSMS (чтобы попасть туда, щелкните сервер правой кнопкой мыши и выберите Properties), а также непосредственно в мастере, как показано на рис. 3.

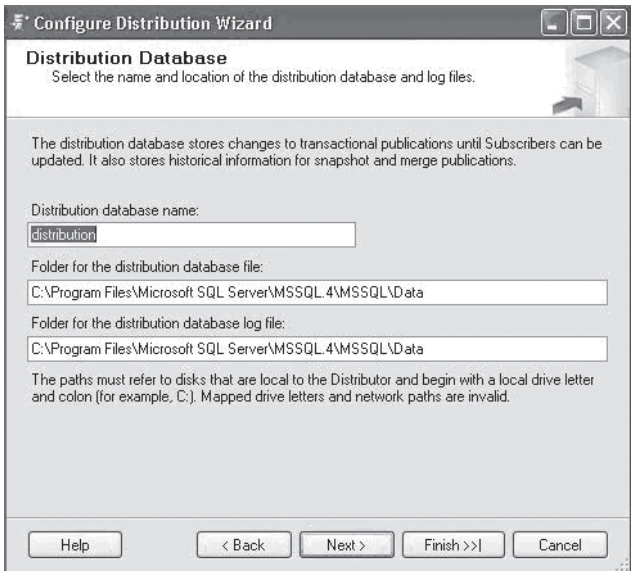


Рис. 3

Следующий экран позволяет настроить серверы на использование текущего дистрибьютора, если они сконфигурированы как издатели (см. рис. 4). Здесь есть пара интересных настроек. Во-первых, если щелкнуть на кнопку с многоточием напротив издателя, откроется диалоговое окно, где можно указать учетные данные агента чтения журнала и папку для снимков для этого издателя, как показано на рис. 5.

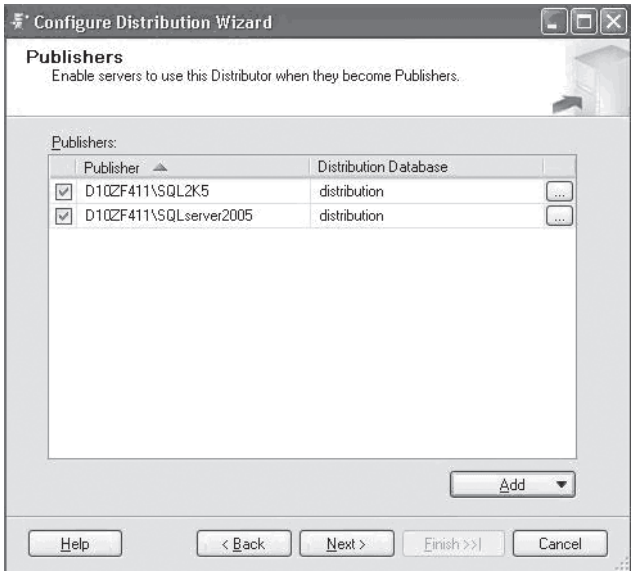


Рис. 4

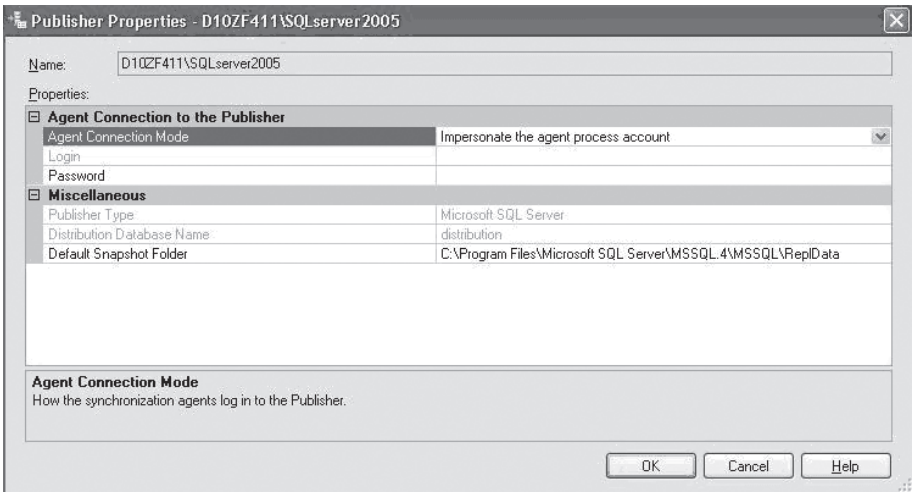


Рис. 5

Во-вторых, кнопка Add позволяет добавить публикующий сервер на базе SQL Server или Oracle. Обратите внимание на эту функцию, потому что в предыдущих версиях нельзя было добавить издателя на базе Oracle.

После активации издателей вы можете установить пароль для удаленных издателей (см. рис. 6). Вы должны ввести пароль дважды. SQL Server 2005 позволяет администратору применять политики паролей и устанавливать для паролей сроки истечения действия. В этом случае мастер предупредит вас, что вводимый пароль должен соответствовать требованиям политики.

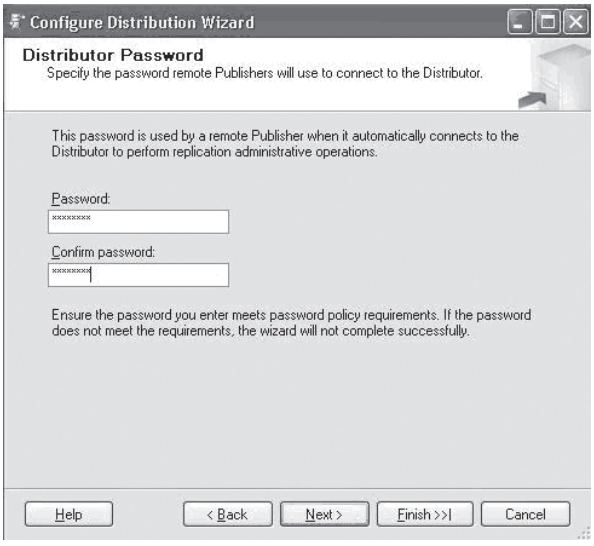


Рис. 6

Щелкнув Next на этом экране, вы можете настроить распределительный сервер прямо сейчас, сохранить сценарий, чтобы запустить его позже, или сделать и то, и другое. Если вы выбрали сохранение сценария, вам потребуется указать место для сохранения файла. На данном этапе мастер выводит краткий список шагов, которые он собирается предпринять. Когда вы щелкните Finish мастер создаст и/или сохранит сценарий для добавления дистрибьютора в зависимости от того, что вы указали.

В листинге 1 приведен сценарий для настройки дистрибьютора и добавления издателя.

Листинг 1. Пример сценария для настройки дистрибьютора и добавления издателя.

```
use master
GO
exec sp_adddistributor @distributor = N'server\instance',
                @password    = N''
GO

/* создание и конфигурирование распределительной базы данных */

exec sp_adddistributiondb @database = N'distribution',
@data_folder = N'C:\Program Files\Microsoft SQL Server\MSSQL.4\MSSQL\Data',
@data_file_size = 4, @log_folder = N'C:\Program Files\Microsoft SQL
Server\MSSQL.4\MSSQL\Data',
@log_file_size = 2, @min_distretention = 0, @max_distretention = 72, @history_retention
= 48,
@security_mode = 1
GO

use [distribution]
if (not exists (select * from sysobjects where name = 'UIProperties' and type = 'U '))
 create table UIProperties(id int)
if (exists
(select * from ::fn_listextendedproperty('SnapshotFolder', 'user', 'dbo', 'table',
'UIProperties', null, null)))

EXEC sp_updateextendedproperty
N'SnapshotFolder', N'C:\Program Files\Microsoft SQL
Server\MSSQL.4\MSSQL\ReplData',
'user', dbo, 'table', 'UIProperties'
else
 EXEC sp_addextendedproperty N'SnapshotFolder',
 'C:\Program Files\Microsoft SQL Server\MSSQL.4\MSSQL\ReplData', 'user', dbo, 'table',
 'UIProperties'
GO

/* добавление издателя */


exec sp_adddistpublisher @publisher = N'server\instance', @distribution_db =
```

```
N'distribution',
@security_mode = 1,
@working_directory = N'C:\Program Files\Microsoft SQL
Server\MSSQL.4\MSSQL\ReplData',
@trusted = N'false', @thirdparty_flag = 0, @publisher_type = N'MSSQLSERVER'
GO
```

После конфигурирования распределительной базы данных вы можете просматривать или изменять свойства дистрибьютора, щелкнув правой кнопкой на папке replication и выбрав Distributor Properties. Откроется диалоговое окно с двумя страницами – general и publishers. Страница general позволяет просмотреть свойства распределительной базы данных и изменить настройки сохранения транзакций и/или истории (см. рис. 7).

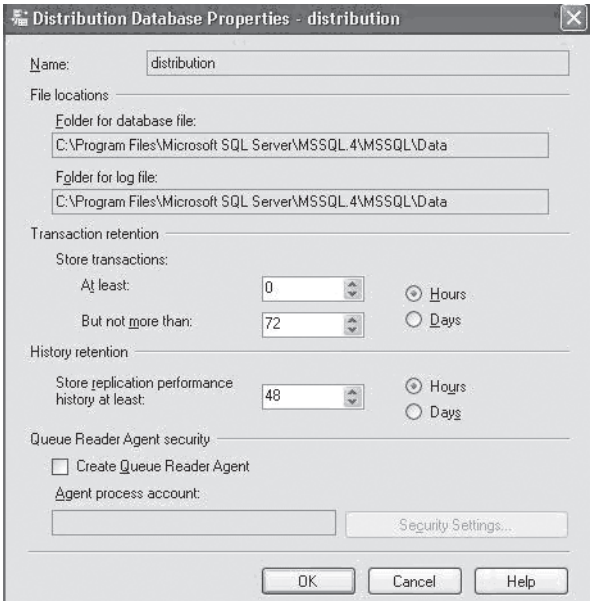


Рис. 7

Обратите внимание, что на этом экране можно также создать и настроить агент чтения очереди. Впрочем, это выходит за рамки этой статьи.

Страница publishers диалогового окна Distribution Database Properties позволяет изменить свойства существующего издателя или добавить нового.

Внимание! Configure Distribution Wizard добавляет на сервер учетную запись distributor_admin, члена серверной роли sysadmin, а значит возможности ее воздействия на сервер практически неограниченны. Именно поэтому абсолютно необходимо применять для подключения к распределительному серверу стойкий пароль.

Создание публикации

После настройки дистрибьютора можно создавать публикации. Для вызова мастера публикаций щелкните правой кнопкой локальную папку publications и выберите в контекстном меню пункт New Publication. Как и у Distribution Configuration Wizard первый экран чисто информативный и его можно пропустить. На втором экране нужно выбрать базу данных, для которой вы хотите создать публикацию, в этой статье я опубликую базу данных AdventureWorksDW, которую можно создать в процессе установки SQL Server 2005. После выбора базы данных вы должны указать тип репликации. Мастер предлагает на выбор четыре варианта:

- Snapshot Publication (Мгновенная публикация)
- Transactional Publication (Транзакционная публикация)
- Transactional Publication with Updatable Subscriptions (Транзакционная публикация с возможностью обновления подписчиков)
- Merge Publication (Публикация слиянием)

Мастер дает краткое описание каждого типа публикации. В моем примере я использую транзакционную публикацию. Узнать о типах публикации больше вы можете из моих предыдущих статей.

Транзакционная публикация может содержать одну или несколько статей (*article*). Статьей может быть таблица, представление (в том числе индексированное), пользовательская функция или хранимая процедура. В этом примере я буду реплицировать таблицу dimAccount из базы данных AdventureWorksDW. Я могу реплицировать все столбцы данной таблицы или некоторое их подмножество, как показано на рис. 8.

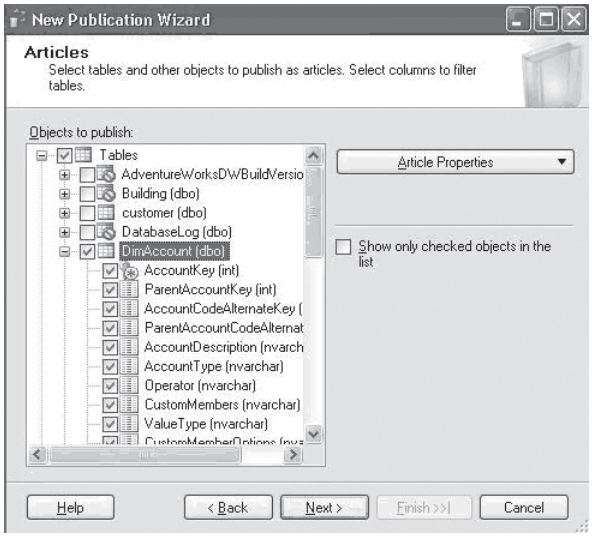


Рис. 8

У репликации есть свои правила, когда дело касается фильтрации столбцов. При транзакционной репликации запрещена фильтрация столбцов первичного ключа. Если разрешено обновление подписчиков, обязательно нужно реплицировать столбец msrepl_tran_version (он добавляется SQL Server при создании такой публикации). Более того, публикации с возможностью обновления подписчиков должны реплицировать все столбцы, которые не являются столбцами identity и при этом в них не разрешены значения null и нет предопределенного значения по умолчанию.

Если поставить флажок Show Only Checked Objects in the List, мастер оставит в списке только отмеченные статьи. С помощью кнопки Article Properties можно установить свойства для выделенной статьи или для всех табличных статей. Как видно на рис. 9, для каждой статьи можно задать значения множества связанных с репликацией свойств.

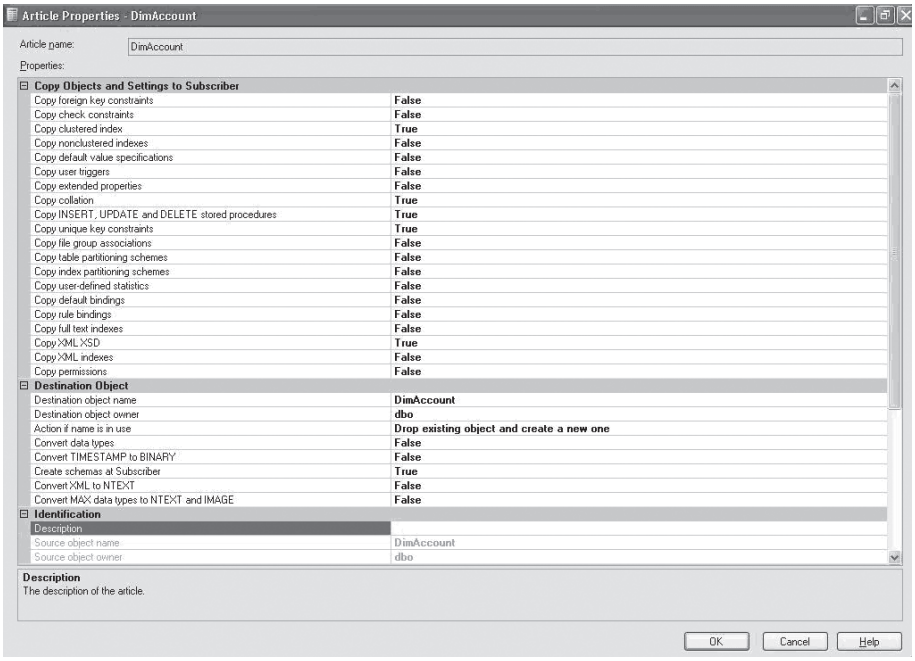


Рис. 9

Большинство свойств, которые можно настроить для табличных статей, не требуют пояснения; например, флажок Copy Foreign Key Constraints указывает, что при создании таблицы в базе данных подписчика нужно добавить ограничения внешнего ключа.

Но несколько свойств заслуживают дополнительных пояснений.

- **Destination Object Name, Destination Object Owner.** Имя и владелец принимающей таблицы и исходного объекта не обязательно должны совпадать.
- **Convert Data Types.** Этот параметр автоматически приводит пользовательский тип данных к базовому, потому что на подписчике этот пользовательский тип данных может не существовать.
- **Convert TIMESTAMP to BINARY.** При репликации столбца типа TIMESTAMP можно преобразовать его значение в BINARY. Тип данных TIMESTAMP отслеживает последовательные модификации; при каждом изменении данных в строке SQL Server автоматически меняет значение столбца типа TIMESTAMP. Это важно, ведь если вы будете недостаточно аккуратны, то можете получить разные значения в таком столбце на издателе и подписчике.
- **Convert MAX Data Types to NTEXT and IMAGE.** Этот параметр приводит появившиеся в SQL Server 2005 типы данных VARCHAR(MAX) и VARBINARY(MAX) к соответствующим типам данных, поддерживаемым в предыдущих версиях.
- **Convert XML to NTEXT.** Приводит новый тип данных XML к NTEXT.
- Еще один параметр, это автоматическое управление диапазоном значений identity. Администратор баз данных может задать диапазоны допустимых значений для столбца identity в базах данных на издателе и подписчике. Например, можно назначить значения от 1,000,000 и выше публикующему серверу, а от 1 до 1,000,000 - подписчику. Когда база данных на издателе достигнет верхней границы диапазона, ей будет автоматически выделен новый таким образом, чтобы значения свойства identity на издателе и подписчике не перекрывались.
- Последняя группа параметров (не показана на рис. 9) определяет, как реплицировать выражения INSERT, UPDATE и DELETE.

После установки необходимых свойств для реплицируемой статьи можно добавить фильтры публикации (см. рис. 10). В предыдущих версиях SQL Server их называли *горизонтальными фильтрами (или горизонтальным разбиением)* – их создают с помощью блока WHERE, чтобы ограничить число публикуемых строк. Выше было показано, что публикации можно фильтровать вертикально, указав, какие столбцы нужно публиковать.

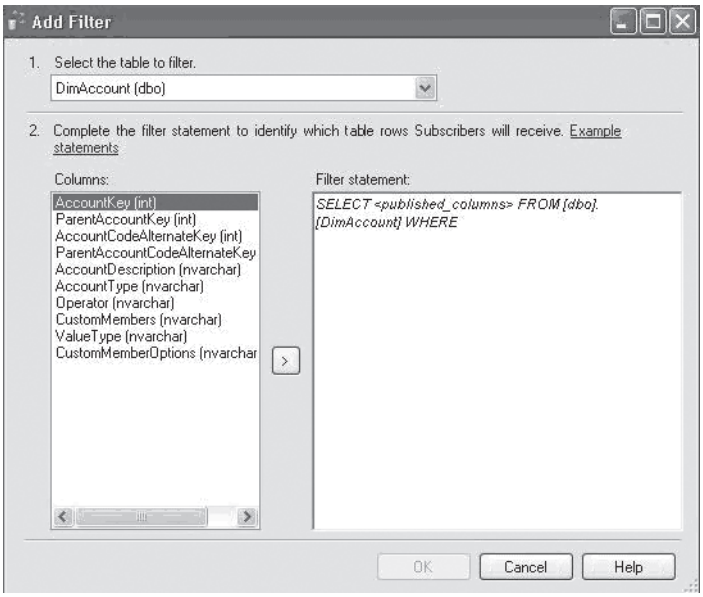


Рис. 10

Следующий шаг – создание мгновенного снимка и/или задание расписания для агента мгновенных снимков, как показано на рис. 11.

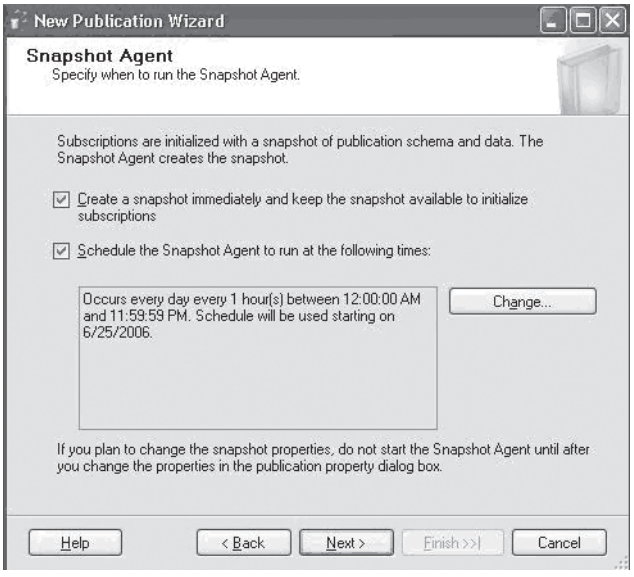


Рис. 11

Агент моментальных снимков копирует схему и данные реплицируемых статей в папку для мгновенных снимков. Если на этом экране щелк-

нуть кнопку Change, откроется стандартное диалоговое окно для назначения заданий; агента можно запускать раз в месяц, раз в неделю, ежедневно и даже несколько раз в день.

Далее нужно указать параметры безопасности для агентов мгновенных снимков и чтения журнала (см. рис. 12). На безопасности репликации я более подробно остановлюсь в одной из следующих статей об агентах транзакционной репликации. Пока что вам достаточно знать, что можно настроить всех агентов разом или использовать для каждого отдельную учетную запись.

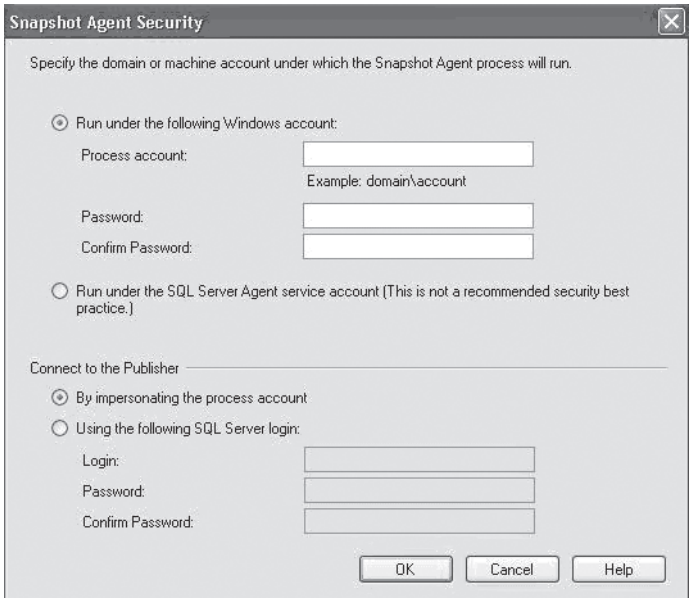


Рис. 12

После этого мастер предложит вам сохранить команды для создания публикации в виде сценария. Просмотрите список шагов, которые мастер намерен предпринять, затем укажите имя публикации и щелкните Finish для ее создания.

В листинге 2 приведен сценарий для создания публикации.

Листинг 2. Пример сценария для создания публикации.

```
use [AdventureWorksDW]
exec sp_replicationdboption @dbname = N'AdventureWorksDW', @optname = N'publish',
@value = N'true'
GO
-- Добавление транзакционной публикации
use [AdventureWorksDW]
exec sp_addpublication @publication = N'DimAccount',
@description = N'Transactional publication of database ''AdventureWorksDW'' from
```

```
Publisher ''server\instance''.',
@sync_method = N'concurrent', @retention = 0, @allow_push = N'true', @allow_pull =
N'true', @allow_anonymous = N'true',
@enabled_for_internet = N'false', @snapshot_in_defaultfolder = N'true',
@compress_snapshot = N'false',
@ftp_port = 21, @ftp_login = N'anonymous', @allow_subscription_copy = N'false',
@add_to_active_directory = N'false',
@repl_freq = N'continuous', @status = N'active', @independent_agent = N'true',
@immediate_sync = N'true',
@allow_sync_tran = N'false', @autogen_sync_procs = N'false', @allow_queued_tran =
N'false', @allow_dts = N'false',
@replicate_ddl = 1, @allow_initialize_from_backup = N'false', @enabled_for_p2p =
N'false',
@enabled_for_het_sub = N'false'
GO


exec sp_addpublication_snapshot @publication = N'DimAccount', @frequency_type = 4,
@frequency_interval = 1,
@frequency_relative_interval = 1, @frequency_recurrence_factor = 0,
@frequency_subday = 8,
@frequency_subday_interval = 1, @active_start_time_of_day = 0,
@active_end_time_of_day = 235959,
@active_start_date = 0, @active_end_date = 0, @job_login = null, @job_password = null,
@publisher_security_mode = 1


use [AdventureWorksDW]
exec sp_addarticle @publication = N'DimAccount', @article = N'DimAccount',
@source_owner = N'dbo',
@source_object = N'DimAccount', @type = N'logbased', @description = null,
@creation_script = null,
@pre_creation_cmd = N'drop', @schema_option = 0x000000000803509F,
@identityrangemanagementoption = N'manual',
@destination_table = N'DimAccount', @destination_owner = N'dbo', @vertical_partition
= N'false',
@ins_cmd = N'CALL sp_MSins_dboDimAccount', @del_cmd = N'CALL
sp_MSdel_dboDimAccount',
@upd_cmd = N'SCALL sp_MSupd_dboDimAccount'
GO
```

Свойства только что созданной публикации можно просмотреть, развернув локальную папку publications, щелкнув нужную публикацию правой кнопкой мыши и выбрав в контекстном меню пункт Properties. У диалогового окна свойств несколько страниц, каждая из которых отвечает конкретной цели.

- **General.** Показывает имя, описание и тип публикации, а также базу данных, на которой она основана. На этой странице можно изменить параметры истечения подписки.

- **Articles.** Позволяет просмотреть опубликованные статьи, изменить их свойства или добавить к публикации новые статьи.
- **Filter Rows.** Позволяет создавать для статей горизонтальные фильтры.
- **Snapshot.** Здесь можно указать местоположение папки для мгновенных снимков, формат снимков и задать дополнительные сценарии, которые будут выполнены до и после создания снимка.
- **FTP Snapshot.** Эти параметры позволяют подписчикам загружать снимки с помощью FTP. Защита FTP настраивается здесь же.
- **Agent Security.** Контролирует параметры безопасности для агентов чтения журнала и мгновенных снимков.
- **Publication Access List.** Задает учетные записи SQL Server и Windows с полномочиями создавать и синхронизировать подписку.
- **Subscription Options.** Содержит множество параметров для подписчиков на текущую публикацию.

В следующей таблице приведены параметры подписки, которые можно установить через диалоговое окно Publication Properties. Обратите внимание, что некоторые из них появились в версии SQL Server 2005.

Параметр подписки	Новшество SQL Server 2005	Описание
Independent Distribution Agent	Нет	Указывает, использовать ли агента, независимого от других публикаций в этой базе данных.
Snapshot Always Available	Да	Делает файлы мгновенных снимков постоянно доступными для инициализации подписки. Этот параметр требует независимого распределительного агента.
Allow Anonymous Subscriptions	Нет	Включает поддержку анонимной подписки, обычно применяется для репликации через Интернет. Требует установки Snapshot Always Available.
Attachable Subscription Database	Нет	Указывает, можно ли создавать подписку присоединением публикуемой базы данных на подписчике. Требует установки Snapshot Always Available.
Allow Pull Subscriptions	Нет	Добавляет поддержку запросной подписки.
Allow Initialization from Backup Files	Да	Позволяет создавать подписку путем восстановления публикуемой базы данных на подписчике из резервной копии.
Allow Non-SQL Server Subscribers	Нет	Разрешает подписчиков на базе отличных от SQL Server СУБД.

Параметр подписки	Новшество SQL Server 2005	Описание
Allow Data Transformations	Нет	Позволяет выполнять преобразования данных перед отправкой их подписчикам. Этот параметр использовать не рекомендуется.
Replicate Schema Changes	Да	Возможно, самое важное из новшеств в версии SQL Server 2005. Если параметр установлен в YES (значение по умолчанию), в процессе репликации подписчикам доставляются любые выполненные на издателе выражения ALTER TABLE. В предыдущих версиях некоторые изменения схемы требовали удаления статей из публикации, что приводило к простоям.
Allow Peer-to-Peer Subscriptions	Да	Определяет, могут ли подписчики устанавливать одноранговые (peer-to-peer) отношения с издателями. Одноранговая топология позволяет нескольким узлам одновременно выполнять функции подписчиков и издателей. Это похоже на двунаправленную репликацию в предыдущих версиях.
Allow Immediate Updating Subscriptions	Нет	Контролирует, доставлять ли подписчикам изменения данных на издателе немедленно после их внесения.
Allow Queued Updating Subscriptions	Нет	Указывает, можно ли помещать изменения данных на подписчиках в очередь, чтобы позже доставить их издателю. Этот параметр может быть полезен, если канал связи между подписчиком и издателем недостаточно надежен.

Создание подписки

В отличие от предыдущих версий в SQL Server 2005 для создания и принудительной (push), и запросной (pull) подписки используется один и тот же мастер. Чтобы его запустить, щелкните правой кнопкой мыши на публикации (или локальной папке subscriptions) и выберите в контекстном меню пункт New Subscriptions. Пропустив первый вступительный экран, выберите публикацию, для которой собираетесь создать подписку. Потом укажите, принудительную или запросную подписку вы хотите использовать (см. рис. 13). Запросная подписка снижает нагрузку на издателя, в то время как принудительная облегчает администрирование, позволяя централизованно управлять всеми подписками. В этом примере я использую принудительную подписку, но в случае запросной подписки экраны мастера почти идентичны.

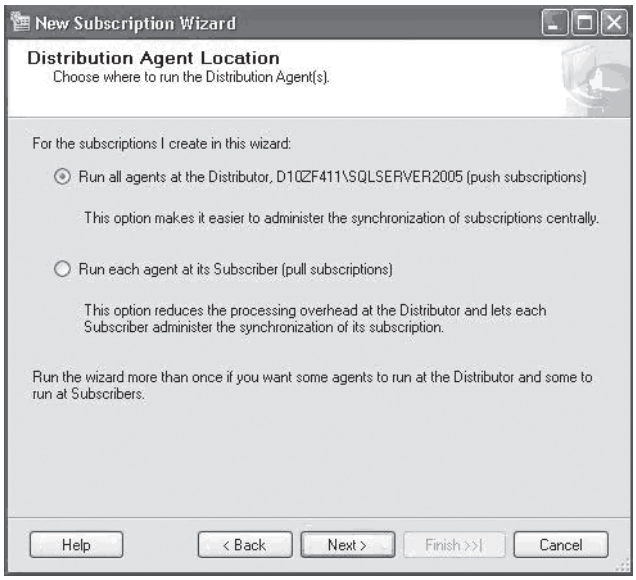


Рис. 13

Далее вы выбираете сервер-подписчик и базу данных на нем, как показано на рис. 14. Можно использовать существующую базу данных или создать новую; если вы предпочтете второй вариант, то получите стандартное диалоговое окно для создания баз данных. Но более интересно то, что мастер позволяет указывать в качестве подписчика сервер с отличной от SQL Server СУБД. Для принудительной подписки можно выбрать подписчика на базе Oracle или IBM DB2, для запросной подписки подходят лишь подписчики на базе SQL Server.

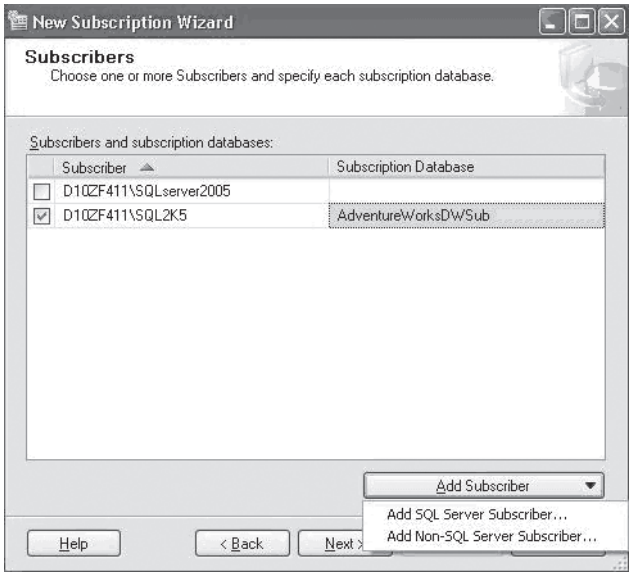


Рис. 14

У подписчиков на базе других систем есть определенные ограничения и особенности, о которых лучше знать заранее. Если вы планируете использовать такую подписку, ознакомьтесь сначала с разделом Heterogeneous Database Replication в SQL Server Books Online.

После задания сервера-подписчика и базы данных подписки нужно настроить параметры безопасности для распределительного агента (см. рис. 15). Я расскажу об этом более подробно в другой статье, а пока просто помните, что у вас есть два варианта: олицетворять SQL Server Agent или отвести для распределительного агента отдельную учетную запись Windows или SQL Server. В этом примере я воспользуюсь учетной записью службы SQL Server Agent для запуска распределительного агента и подключения к подписчику.

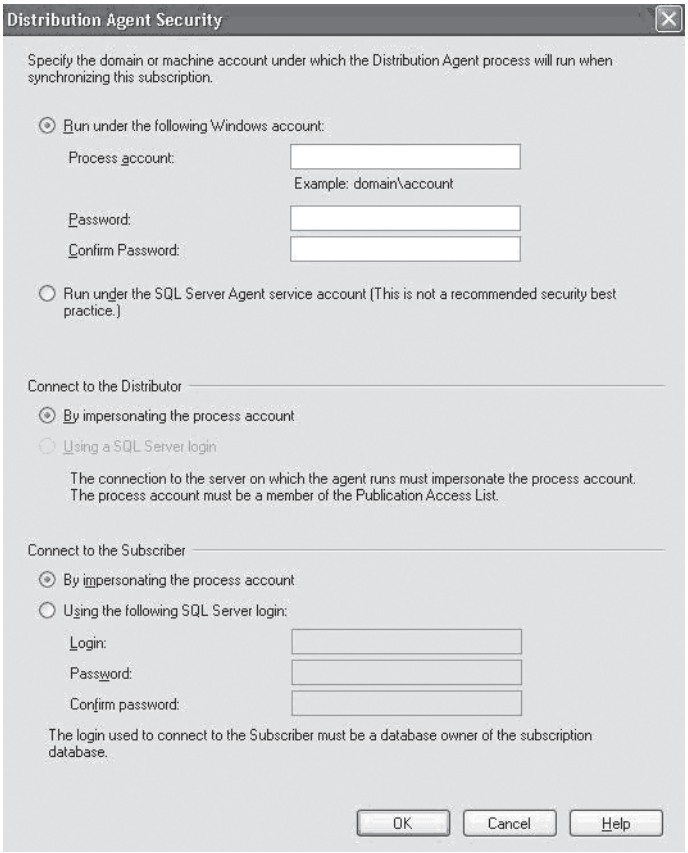


Рис. 15

Теперь пора задать расписание синхронизации – как часто отправлять реплицированные транзакции подписчикам. Если вы хотите добиться минимальной задержки, то лучшим вариантом будет *непрерывная* репликация транзакций, однако это увеличивает нагрузку на издателя

в случае принудительной подписки и на подписчика в случае запросной. *Доставка по расписанию* – хороший выбор, если вы хотите минимизировать нагрузку в рабочие часы и доставлять команды каждый день в определенное время. Доставка по требованию вполне подойдет, если вы собираетесь синхронизировать базы данных лишь время от времени.

После задания желаемого расписания синхронизации вы можете инициализировать базу данных подписки (см. рис. 16). В процессе инициализации происходит создание схем публикуемых объектов и копирование данных из папки для мгновенных снимков в базу данных подписки, кроме того, в этой базе данных создаются хранимые процедуры для репликации. В диалоговом окне можно указать, что вы не хотите инициализировать подписку – это полезно, если схема и данные уже присутствуют на подписчике. Еще один вариант – инициализировать подписку при первой синхронизации, то есть при первом запуске агента мгновенных снимков.

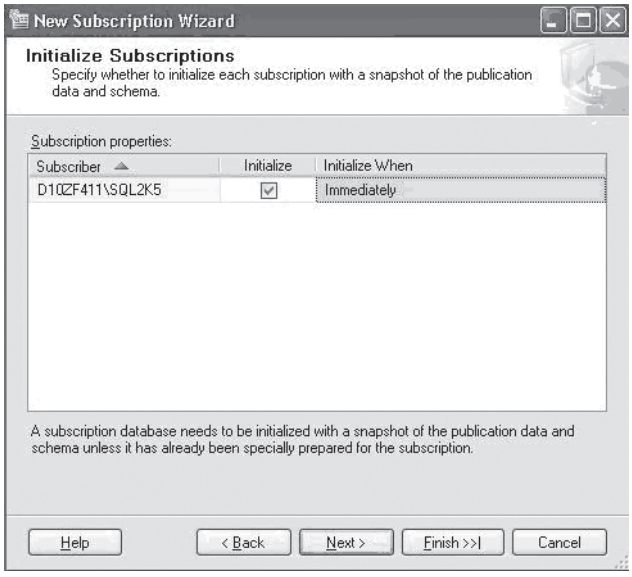


Рис. 16

Теперь вы ввели всю необходимую информацию. На данном этапе можно создать подписку и/или сохранить сценарий для ее создания. Мастер позволяет просмотреть список действий, которые будут выполнены после нажатия кнопки Finish.

В листинге 3 приведен сценарий для создания подписки.

Листинг 3. Пример сценария для создания подписки.

```
----------НАЧАЛО: Сценарий для запуска на издателе---------
use [AdventureWorksDW]
GO
```

```
exec sp_addsubscription @publication = N'DimAccount',
@subscriber = N'server\subscriber_instance',
@destination_db = N'AdventureWorksDWSub',
@subscription_type = N'Push',
@sync_type = N'automatic',
@article = N'all',
@update_mode = N'read only',
@subscriber_type = 0

exec sp_addpushsubscription_agent @publication = N'DimAccount',
@subscriber = N'server\subscriber_instance',
@subscriber_db = N'AdventureWorksDWSub',
@job_login = null,
@job_password = null,
@subscriber_security_mode = 1,
@frequency_type = 64,
@frequency_interval = 0,
@frequency_relative_interval = 0,
@frequency_recurrence_factor = 0,
@frequency_subday = 0,
@frequency_subday_interval = 0,
@active_start_time_of_day = 0,
@active_end_time_of_day = 235959,
@active_start_date = 20060627,
@active_end_date = 99991231,
@enabled_for_syncmgr = N'False',
@dts_package_location = N'Distributor'
```

Чтобы просмотреть свойства подписки, разверните нужную публикацию в локальной папке publications, щелкните подписку правой кнопкой мыши и выберите в контекстном меню пункт Properties. На рис. 17 приведено окно свойств для моего примера.

Если вы заглянете в папку stored procedures в базе данных подписки, то обнаружите там три новых процедуры, которые применяются для доставки реплицированных транзакций:

- sp_MSupd_dboDimAccount;
- sp_MSdel_dboDimAccount;
- sp_MSins_dboDimAccount.

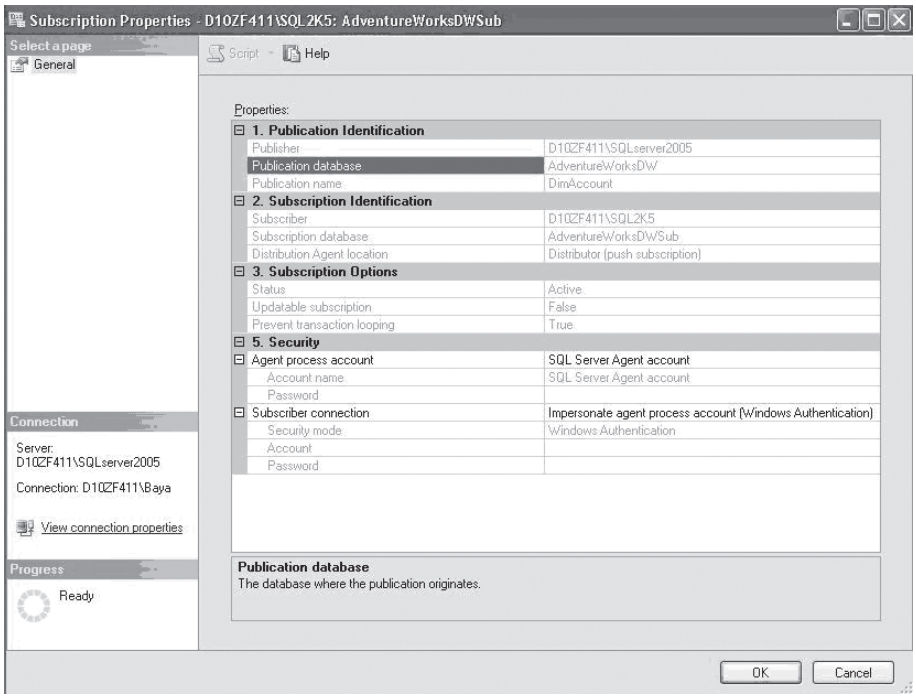


Рис. 17

Тестирование репликации

Теперь, когда мы настроили репликацию, можно ее протестировать, выполнив несколько команд SQL. Я выполню на издателе следующее выражение UPDATE:

```
UPDATE dimAccount
SET AccountDescription = 'work in progress'
WHERE AccountKey = 11
```

SQL Server сообщил мне, что запрос затронул одну строку. Далее я переключаю контекст на распределительную базу данных и запускаю хранимую процедуру sp_browsereplcmds. SQL Server показывает, что в процессе доставки находятся следующие команды:

```
{CALL [sp_MSupd_dboDimAccount] (,,,,N'work in progress',,,,,,11,0x1000)}
```

Наконец, я посылаю запрос к базе данных подписки, чтобы убедиться, что внесенные на издателе изменения были успешно доставлены подписчику:

```
SELECT
 AccountDescription
FROM dimAccount
WHERE AccountKey = 11
```

Результат:

```
AccountDescription
--------------------------
work in progress
```

Заключение

В этой статье я показал, как настроить транзакционную репликацию в SQL Server 2005. Мы сконцентрировались на том, как сконфигурировать дистрибьютора, создать публикацию и на нее подписаться. Приведенный в статье пример был очень упрощенным. В реальной ситуации при настройке репликации, скорее всего, возникнут некоторые сложности, но этот обзор даст вам достаточное начальное представление об общей картине репликации с помощью SQL Server 2005.

Следующие статьи из этой серии расскажут о транзакционной репликации более детально. Я покажу, как обслуживать репликацию и решать возникающие проблемы, а также как пользоваться некоторыми продвинутыми функциями.

Обслуживание транзакционной репликации в SQL Server 2005

По материалам статьи Байя Павлиашвили (Baya Pavliashvili) «Maintaining Transactional Replication with SQL Server 2005»

Перевод предоставлен Дмитрием Артемовым

В SQL Server 2005 Microsoft представила несколько замечательных улучшений в области репликации, снимающих некоторые проблемы обслуживания, возникавшие в предыдущих версиях. Байя Павлиашвили демонстрирует шаги, необходимые для поддержки типичной публикации, и объясняет, как новые параметры доставки изменений данных могут помочь подогнать схему репликации под нужды именно ваших приложений.

Первая статья этой серии представляла собой введение в настройку репликации в SQL Server 2005. В простых сценариях настройка репликации не представляет особой сложности, однако репликация требует обслуживания. В этой статье я покажу вам, как обслуживать хранимые процедуры для репликации, вносить изменения в существующие публикации и повторно инициализировать подписку в приложении к SQL Server 2005. Как и в первой статье, я подчеркну наиболее существенные различия между последней и предыдущими версиями SQL Server.

Хранимые процедуры для репликации

По умолчанию при репликации выполненные на издателе команды INSERT, UPDATE и DELETE замещаются на подписчике соответствующими хранимыми процедурами. Это рекомендованная практика, поскольку хранимые процедуры разбивают большие команды INSERT, UPDATE и DELETE на более мелкие части – за раз обрабатывается лишь одна строка. Так что выражение UPDATE, которое влияет на 1000 строк, будет преобразовано в 1000 вызовов хранимой процедуры для обновления. Изменять по одной строке за раз не слишком эффективно, но это снижает негативное влияние блокировок на подписчике, снижая количество проблем при одновременном доступе к данным.

В отличие от предыдущих версий, в SQL Server 2005 есть множество

новых параметров, касающихся доставки выражений, что позволяет настроить репликацию в соответствии с вашими нуждами. Эти параметры кратко поясняются в следующей таблице:

Выражение	Параметр	Значение	Новшество SQL Server 2005
INSERT	Call stored procedure (default)	Выполненные на издателе выражения INSERT преобразуются в вызовы хранимых процедур, которые вставляют в базу по одной строке за раз.	Нет
INSERT	Do not replicate INSERT statements	Выполненные на издателе выражения INSERT не пересылаются подписчикам.	Да
INSERT	INSERT statement	На издателе и подписчике выполняются одинаковые выражения INSERT.	Нет

INSERT	INSERT statement without column list	При репликации выражения INSERT не указывается список столбцов:
		`insert into [dbo].[DimAccount]`
		`values (11,9,1164,1160,N'work in progress', N'Assets',N'+',NULL,N' Currency',NULL)`
			Да

UPDATE	Do not replicate UPDATE statements	Выполненные на издателе выражения UPDATE не пересылаются подписчикам.	Да
UPDATE	UPDATE statement	На издателе и подписчике выполняются одинаковые выражения UPDATE.	Нет
UPDATE	CALL stored procedure (default)	Выполненные на издателе выражения UPDATE преобразуются в вызовы хранимых процедур, которые изменяют по одной строке в базе за раз.	Нет

UPDATE	MCALL stored procedure	Выполненные на издателе выражения UPDATE преобразуются в вызовы хранимых процедур, которые изменяют по одной строке в базе за раз. При этом для изменяемых столбцов передаются новые значения, а для оставшихся неизменными – NULL, как показано здесь:
		`{CALL [sp_MSupd_dboDimAccount]`
		`(NULL,NULL,NULL,NULL,N'work in process',NULL,NULL,NULL,NULL,NULL,11, 0x1000)}`
			Нет

Выражение	Параметр	Значение	Новшество SQL Server 2005
UPDATE	XCALL stored procedure	Выполненные на издателе выражения UPDATE преобразуются в вызовы хранимых процедур, которые изменяют по одной строке в базе за раз. При этом передаются значения всех столбцов, как показано здесь: {CALL [sp_MSupd_dboDimAccount] (11, 9, 1164, 1160, N'work in process', N'Assets', N'+', NULL, N'Currency', NULL, 11, 9, 1164, 1160, N'work in progress', N'Assets', N'+', NULL, N'Currency', NULL)}	Нет
UPDATE	SCALL stored procedure	Выполненные на издателе выражения UPDATE преобразуются в вызовы хранимых процедур, которые изменяют по одной строке в базе за раз. При этом для изменяемых столбцов передаются новые значения, а для оставшихся неизменными не передается ничего, как показано здесь: {CALL [sp_MSupd_dboDimAccount] (,,,,N'work in process',,,,,,11,0x1000)}	Да
DELETE	CALL stored procedure (default)	Выполненные на издателе выражения DELETE преобразуются в вызовы хранимых процедур, которые удаляют по одной строке в базе за раз.	Нет
DELETE	Do not replicate DELETE statements	Выполненные на издателе выражения DELETE не пересылаются подписчикам.	Да
DELETE	DELETE statement	На издателе и подписчике выполняются одинаковые выражения DELETE.	Нет
DELETE	XCALL stored procedure	Выполненные на издателе выражения DELETE преобразуются в вызовы хранимых процедур, которые удаляют по одной строке в базе за раз. При этом передаются значения всех столбцов, как показано здесь: {CALL [sp_MSdel_dboDimAccount] (11,9,1164,1160, N'work in progress',N'Assets',N'+',NULL, N'Currency',NULL)}	Нет

Обратите внимание, что хотя в предыдущих версиях нельзя было проигнорировать команды INSERT, UPDATE или DELETE, вы могли модифицировать созданные на подписчике хранимые процедуры для репликации, чтобы добиться точно такого же эффекта. Впрочем, делать этого не рекомендуется, потому что измененные репликационные процедуры

не поддерживаются. Но что более важно, даже если вы измените процедуры для репликации, команды по-прежнему будут доставляться в распределительную базу данных – они просто не будут приводить к изменениям на подписчике. В SQL Server 2005 если вы укажете не реплицировать команды INSERT, UPDATE или DELETE, то они вообще не будут отправлены дистрибьютору. Более того, на подписчиках НЕ создаются соответствующие хранимые процедуры.

Текст репликационных хранимых процедур, выполняемых на подписчике, изменился мало; каркас процедур для выражений INSERT, UPDATE и DELETE приведен ниже:

```
/*
** процедура DELETE
** в качестве единственного параметра передается первичный ключ
*/
create procedure [dbo].[sp_MSdel_dboDimAccount]
@pkc1 int
as
begin
delete [dbo].[DimAccount] where [AccountKey] = @pkc1

if @@rowcount = 0
if @@microsoftversion>0x07320000
exec sp_MSreplraiserror 20598
end
```

Примечание. Если вы применяете для выражений DELETE параметр XCALL, хранимая процедура будет принимать параметры для каждого столбца.

```
/*
** процедура INSERT
*/
create procedure [dbo].[sp_MSins_dboDimAccount]
 @c1 int,@c2 int,@c3 int,@c4 int,@c5 nvarchar(50),@c6 nvarchar(50),@c7
nvarchar(50),@c8 nvarchar(300),@c9 nvarchar(50),@c10 nvarchar(200)
as
begin
insert into "dbo"."DimAccount"(
 "AccountKey"
,"ParentAccountKey"
,"AccountCodeAlternateKey"
,"ParentAccountCodeAlternateKey"
,"AccountDescription"
,"AccountType"
,"Operator"
,"CustomMembers"
,"ValueType"
```

```
,"CustomMemberOptions"
 )
values (
 @c1
,@c2
,@c3
,@c4
,@c5
,@c6
,@c7
,@c8
,@c9
,@c10
 )
end

/*
** процедура UPDATE
*/

create procedure [dbo].[sp_MSupd_dboDimAccount]
 @c1 int = null,@c2 int = null,@c3 int = null,@c4 int = null,@c5 nvarchar(50) =
null,@c6 nvarchar(50) = null,@c7 nvarchar(50) = null,@c8 nvarchar(300) = null,@c9
nvarchar(50) = null,@c10 nvarchar(200) = null,@pkc1 int
,@bitmap binary(2)
as
begin
update "dbo"."DimAccount" set
 "ParentAccountKey" = case substring(@bitmap,1,1) & 2 when 2 then @c2 else
"ParentAccountKey" end
,"AccountCodeAlternateKey" = case substring(@bitmap,1,1) & 4 when 4 then @c3 else
"AccountCodeAlternateKey" end
,"ParentAccountCodeAlternateKey" = case substring(@bitmap,1,1) & 8 when 8 then @c4
else "ParentAccountCodeAlternateKey" end
,"AccountDescription" = case substring(@bitmap,1,1) & 16 when 16 then @c5 else
"AccountDescription" end
,"AccountType" = case substring(@bitmap,1,1) & 32 when 32 then @c6 else
"AccountType" end
,"Operator" = case substring(@bitmap,1,1) & 64 when 64 then @c7 else "Operator" end
,"CustomMembers" = case substring(@bitmap,1,1) & 128 when 128 then @c8 else
"CustomMembers" end
,"ValueType" = case substring(@bitmap,2,1) & 1 when 1 then @c9 else "ValueType" end
,"CustomMemberOptions" = case substring(@bitmap,2,1) & 2 when 2 then @c10 else
"CustomMemberOptions" end
where "AccountKey" = @pkc1

if @@rowcount = 0
 if @@microsoftversion>0x07320000
  exec sp_MSreplraiserror 20598
end
```

Если вы вносите в публикацию изменения, например, добавляете столбцы, изменяете типы данных столбцов или фильтрацию реплицируемых столбцов, вам также придется модифицировать хранимые процедуры для репликации. Запустите Snapshot Agent, и он создаст для вас обновленные хранимые процедуры автоматически; однако, такая роскошь доступна не всегда. Создание мгновенных снимков для публикаций, поддерживающих очень большие базы данных, может потребовать много времени, столько же потребует и применение снимков на подписчике. Если вы не можете себе позволить длительные простои, и не нужно вносить изменения в данные на подписчике, то для обновления хранимых процедур есть более эффективный метод. Можно выполнить на публикующем сервере системную хранимую процедуру sp_scriptpublicationcustomprocs. Эта процедура принимает в качестве единственного параметра имя публикации, вот пример ее вызова:

```
EXECUTE sp_scriptpublicationcustomprocs 'publicationName'
```

В результате вы получите все хранимые процедуры INSERT, UPDATE и DELETE, которые вам нужны на подписчиках.

Если вы используете подписчиков с немедленным обновлением, то можете воспользоваться другой системной хранимой процедурой для получения кода триггеров, создаваемых на таблицах подписчиков. Процедура sp_script_synctran_commands принимает два параметра: имя публикации и имя статьи. Например, следующая команда позволяет получить код триггеров для всех статей:

```
EXEC sp_script_synctran_commands
@publication = 'publicationName', @article = 'all'
```

Изменение схем опубликованных таблиц

Одним из основных ограничений при репликации в предшествующих 2005 версиях SQL Server был неизбежный простой сервера при внесении изменений в схему реплицируемой таблицы. К счастью, SQL Server 2005 позволяет вносить в схему практически любые изменения, не требуя при этом удаления таблицы из публикации. Репликация изменений схем таблиц – возможно, самое существенное и полезное улучшение в области репликации, которое по-настоящему делает SQL Server 2005 СУБД уровня предприятия.

В SQL Server 2000 для добавления/удаления столбцов из реплицируемых таблиц существовали системные процедуры sp_repladdcolumn и sp_repldropcolumn. Однако если вы меняли схему таблицы посредством этих процедур, то приходилось ждать запуска агента моментальных снимков, создавать копию новой схемы и данных, а потом снова ждать, пока распределительный агент применит снимок на подписчике. Конечно же, во время применения снимка на подписчике подверженные этой операции столбцы были недоступны для запросов или модификации –

соответственно, вы получали простой. Вполне пригодной к использованию альтернативой sp_repladdcolumn было применение команды ALTER TABLE для добавления столбца в реплицируемую таблицу и на издателе, и на подписчике. После этого приходилось вручную добавлять столбец к публикации и изменять репликационные хранимые процедуры. Точно так же можно было отфильтровать столбец из существующей публикации и потом выполнить на публикующем сервере выражение ALTER TABLE DROP COLUMN. Но вам все равно пришлось бы ждать, пока агент мгновенных снимков внесет изменения на подписчике, так что без простоя и в этом случае было не обойтись.

Давайте теперь посмотрим, как добавлять или удалять столбцы из опубликованной таблицы в SQL Server 2005. Для начала я выполню в базе AdventureWorksDW на издателе следующее выражение:

```
ALTER TABLE dimAccount ADD test_column VARCHAR(250)
```

Это выражение добавляет новый столбец к опубликованной таблице. Теперь я выполняю системную процедуру sp_browsereplcmds в распределительной базе данных, после чего эта же команда будет отправлена подписчику:

```
ALTER TABLE [dbo].[DimAccount] ADD test_column VARCHAR(250)
```

Кроме того, если я взгляну на репликационные хранимые процедуры на подписчике, то найду ссылки на добавленный столбец:

```
Create procedure [dbo].[sp_MSupd_dboDimAccount]
@c1 int,@c2 int,@c3 int,@c4 int,@c5 nvarchar(50),@c6 nvarchar(50),@c7 nvarchar(50),
@c8 nvarchar(300),@c9 nvarchar(50),@c10 nvarchar(200),
@c11 varchar(250),@pkc1 int ,
@bitmap binary(2)
as
begin

update [dbo].[DimAccount]
set
/* обновление существующих столбцов
** пропущено для экономии места
*/
,[test_column] = case substring(@bitmap,2,1) & 4 when 4 then @c11 else
[test_column] end
where
[AccountKey] = @pkc1
if @@rowcount = 0
if @@microsoftversion>0x07320000
exec sp_MSreplraiserror 20598
end
```

Точно так же, чтобы удалить столбец из реплицируемой таблицы, я могу выполнить следующее выражение:

```
ALTER TABLE dimAccount
DROP COLUMN test_column1
```

Репликация доставит эту команду подписчику, как и при добавлении столбца. SQL Server 2005 незаметно для нас выполняет те же самые операции, которые в предыдущих версиях приходилось делать вручную – после изменения схемы он создает репликационные хранимые процедуры и применяет их к базам данных подписчика. Хотя внесение изменений в схему больше не требует простоя, этой функциональностью не стоит злоупотреблять. Если в распределительной базе данных миллионы транзакций ждут доставки подписчикам, то вряд ли изменения схемы будут реплицированы мгновенно. Изменение схемы таблицы обрабатывается как любая другая транзакция, так что если вы хотите, чтобы схемы на публикующем сервере и подписчике соответствовали друг другу, постарайтесь вносить изменения в структуру таблиц в моменты минимальной активности пользователей.

Возможно, вас интересует, есть ли способ отключить репликацию изменений схемы. Такой способ есть: это можно сделать в диалоговом окне Subscription Properties.

Имейте в виду, чтобы изменения схемы реплицировались, их нужно вносить посредством выражения ALTER TABLE, графическое средство для модификации таблиц в SQL Server Management Studio (SSMS) не подойдет. Если вы попытаетесь изменить реплицируемую таблицу с помощью этого инструмента, SSMS предупредит вас, что таблица является реплицируемой и должна быть удалена из публикации перед внесением изменений в ее схему.

Следующие изменения схем реплицируемых таблиц обнаруживаются автоматически и пересылаются подписчикам:

- **Добавление столбцов.**

  Помните, что нельзя добавить в реплицируемую таблицу столбец со свойством identity.

- **Удаление столбцов.**

  Нельзя удалить столбец (столбцы) первичного ключа.

  Нельзя удалить столбец, добавленный посредством репликации для подписчиков с немедленным обновлением.

- **Изменение типа данных, имени, длины, точности столбца.**

  Здесь тоже разрешены не все изменения, но это ограничение не является специфичным для реплицируемых столбцов. Например, нельзя поменять тип данных столбца с VARCHAR на INT, если в таблице уже есть строковые значения. Вам придется удалить столбец и снова добавить его уже с новым типом данных.

  Нельзя изменять столбцы первичного ключа.

- **Добавление и удаление внешних ключей; при репликации автоматически определяется, есть ли на подписчике таблица, на которую ссылается внешний ключ.**

  Помните, что сконфигурировать репликацию ограничений внешнего ключа и другие параметры, имеющие отношение к схеме, можно в диалоговом окне Article Properties. Если вы внесете изменения в этом окне после создания подписки, то нужно создать новый мгновенный снимок, чтобы изменения вступили в силу.

  На подписчике внешние ключи добавляются с параметром NOT FOR REPLICATION.

- **Применение выражения ALTER TRIGGER; имейте в виду, что CREATE TRIGGER и DROP TRIGGER не будут реплицированы до тех пор, пока новый мгновенный снимок не будет доставлен подписчикам.**

  Определенные команды не реплицируются:

- **Добавление и удаление индексов.**

  Это не слишком большая проблема, поскольку на издателе и подписчике вам могут понадобиться разные индексы.

- **Ограничения, которые включают недетерминистические функции.**

  Чтобы добавить столбец, не внося его в публикацию (вертикальное разделение), выключите репликацию изменений схемы.

Добавление и удаление статей из публикаций

Добавить статьи в публикацию можно в любое время с помощью страницы articles диалогового окна Publication Properties. Помните, что только что добавленные статьи не попадут к подписчикам до тех пор, пока не будет запущен Snapshot Agent. Но вы все равно можете изменять данные в реплицируемой таблице, не дожидаясь этого момента, изменения будут накапливаться в распределительной базе данных, пока не будет создана таблица на подписчике.

С помощью той же страницы диалогового окна Publication Properties можно в любой момент удалить статью из публикации. Однако удаление статьи потребует повторной инициализации подписчиков с помощью нового снимка. При удалении статьи SSMS предупредит вас, что все подписчики на текущую публикацию должны быть инициализированы повторно. Если в процессе настройки репликации вы не выполняли инициализацию подписчиков, то новый мгновенный снимок не создается.

В предыдущих версиях SQL Server перед удалением любой статьи из публикации приходилось удалять подписку.

Если вы предпочитаете добавлять и удалять статьи с помощью сценариев, то можете воспользоваться системными хранимыми процедурами sp_addarticle и sp_droparticle соответственно. Например, следующий сценарий добавляет к публикации статью DimAccount:

```
EXEC sp_addarticle
@publication = N'PublicationName',
@article = N'DimAccount',
@source_owner = N'dbo',
@source_object = N'DimAccount',
@type = N'logbased',
@description = N'',
@creation_script = N'',
@pre_creation_cmd = N'drop',
@schema_option = 0x000000000803519F,
@identityrangemanagementoption = N'manual',
@destination_table = N'DimAccount',
@destination_owner = N'dbo',
@status = 16,
@vertical_partition = N'false',
@ins_cmd = N'SQL',
@del_cmd = N'CALL [sp_MSdel_dboDimAccount]',
@upd_cmd = N'MCALL [sp_MSupd_dboDimAccount]'
```

А эта команда удаляет из публикации статью dimCurrency:

```
EXEC sp_droparticle
@publication= 'PublicationName',
@article= 'dimCurrency',
@force_invalidate_snapshot= 0
```

Обратите внимание, что sp_droparticle не позволяет удалять из публикации статью, на которую существует подписка. Если для этой цели вы воспользуетесь SSMS, то SQL Server удалит подписку, удалит статью, а потом снова добавить подписку на публикацию.

Истечение и деактивация подписки

Истечение и деактивация подписки работают точно так же, как в прошлых версиях. (Если вам нужно освежить свои знания по этому вопросу, обратитесь к моей предыдущей статье «Настройка транзакционной репликации в SQL Server 2005»).

Можно сконфигурировать публикацию так, чтобы подписка не истекала никогда; в этом случае подписка никогда не удаляется, но может быть деактивирована. С другой стороны, можно настроить SQL Server так, чтобы подписка истекала и была удалена, если она не была синхронизирована в заданный промежуток времени (от 1 до 336 часов). Истечение и деактивацию подписки можно настроить на странице general диалога Publication Properties.

Истечением и деактивацией подписки управляет задание по очистке истекшей подписки (по умолчанию запускается один раз в день). Имейте в виду, что задание по очистке дистрибьютора также проверяет, требуется ли деактивация или удаление какой-либо подписки, но основное его назначение – удалять доставленные команды из распределительной базы данных (по умолчанию запускается каждые 10 минут).

При истечении подписки ее можно инициализировать повторно, щелкнув подписку правой кнопкой мыши и выбрав соответствующий пункт в контекстном меню. При повторной инициализации подписки SQL Server генерирует свежий моментальный снимок и применяет его к базе (или базам) данных подписки.

Заключение

В этой статье был приведен обзор операций по обслуживанию транзакционной репликации. SQL Server 2005 значительно упрощает поддержку репликации по сравнению с предыдущими версиями, поскольку большинство изменений схемы можно реплицировать без простоя сервера. Вдобавок к этому, новая версия дает большую гибкость при выборе способа репликации изменений данных.

Следующая статья этой серии будет посвящена мониторингу и решению проблем при репликации в SQL Server 2005. Оставайтесь с нами!

Мониторинг и решение проблем при репликации в SQL Server 2005

По материалам статьи Байя Павлиашвили (Baya Pavliashvili) «Monitoring and Troubleshooting Transactional Replication»

Перевод предоставлен Дмитрием Артемовым

Replication Monitor – прекрасный инструмент для мониторинга и решения проблем в SQL Server 2005. В этой статье Байя Павлиашвили демонстрирует на реальных примерах существенные улучшения средств мониторинга репликации в SQL Server 2005 по сравнению с предыдущими версиями.

Первые две статьи из этой сети были посвящены настройке и обслуживанию транзакционной репликации в SQL Server 2005. В последней версии SQL Server были представлены многочисленные улучшения в области репликации, сильно изменились и средства мониторинга. В этой статье я покажу, как проводить мониторинг репликации и что делать при возникновении проблем.

Встречайте: Replication Monitor 2005

Мониторинг репликации преследует следующие цели:

- Убедиться, что репликация протекает нормально и транзакции доставляются подписчикам.
- Убедиться, что производительность репликации находится на приемлемом уровне, то есть транзакции доставляются за разумное время, так что данные на подписчике соответствуют данным на издателе.
- Выявить и устранить любые ошибки или проблемы с производительностью.

Microsoft великолепно справилась с задачей, воплотив все эти требования в SQL Server 2005; новый набор инструментов позволяет сравнительно легко выполнять все эти операции. Однако чтобы к нему привыкнуть, нужно время.

Чтобы вызвать Replication Monitor, щелкните правой кнопкой мыши папку replication (или локальную папку publications) в SQL Server Management Studio (SSMS) и выберите пункт Launch Replication Monitor в контекстном меню. В предыдущих версиях Replication Monitor запус-

кался в том же окне, что и Enterprise Manager; в SQL Server 2005 он запускается в отдельном окне. Должен предупредить, что окно Replication Monitor может открываться с задержкой, особенно на медленных и сильно загруженных серверах. Replication Monitor – важный инструмент для решения проблем репликации, так что я надеюсь, что этот незначительный недостаток скоро исправят. Хорошая новость заключается в том, что улучшения в области репликации в SQL Server 2005 значительно перевешивают проблемы, с которыми вы можете столкнуться на серверах с низкой производительностью.

У Replication Monitor две панели: в левой панели находится список издателей и ассоциированных с ними публикаций, в правой отображается статус каждой подписки на публикации выбранного сервера. Если вы не нашли в списке нужный публикующий сервер, его можно добавить, щелкнув правой кнопкой мыши узел My Publishers и выбрав Add. Откроется диалоговое окно, где можно создать новую группу мониторинга или добавить/удалить издателя на базе SQL Server или Oracle (как показано на следующем рисунке).

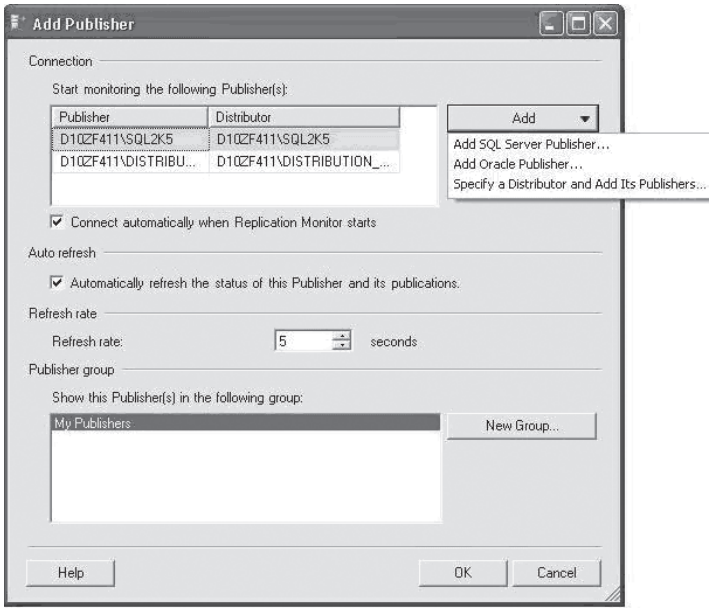


Рис. 1

Replication Monitor позволяет следить за публикациями и подпиской на системах с предыдущими версиями SQL Server, но новые функции доступны только для SQL Server 2005.

Вкладка default в Replication Monitor сообщает, происходит ли в данный момент синхронизация каких-либо подписчиков, и показывает среднее и наихудшее значения производительности этого процесса. О мониторинге производительности репликации вы узнаете позже, а пока

давайте кратко познакомимся с первым экраном, где отображаются все публикации, ассоциированные с текущим издателем (см. рис. 2).

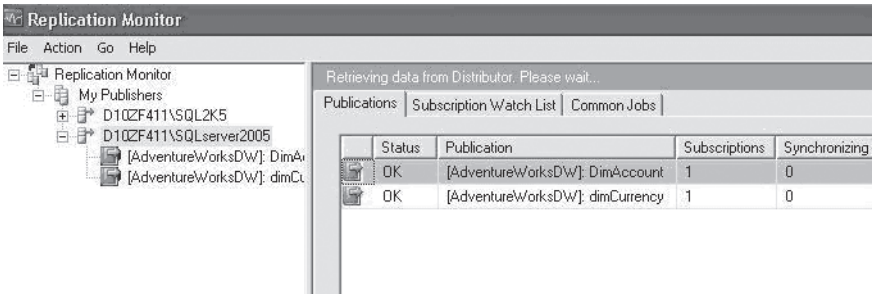


Рис. 2

Если щелкнуть какую-нибудь публикацию в левой панели, в правой появятся данные, специфичные именно для этой публикации; по умолчанию отображаются данные обо всех публикациях. Если у вас десятки подписок, то вкладка Subscription Watch List на правой панели станет вашим лучшим другом – она позволяет фильтровать список, ограничивая его только теми подписками, которые вас интересуют. Для этого на вкладке есть два раскрывающихся списка: первый позволяет указать тип подписки (моментальным снимком, транзакционная или слиянием), а второй связан с производительностью и позволяет выбрать один из следующих вариантов:

- Показать 25 подписок с наихудшей производительностью
- Показать 50 подписок с наихудшей производительностью
- Показать только ошибки и предупреждения
- Показать только ошибки
- Показать только предупреждения
- Показать все работающие подписки (не остановленные из-за ошибки)
- Показать все неработающие подписки

На вкладке Common Jobs можно запускать, останавливать и обновлять задания репликации для всех подписок, как показано на следующем рисунке:

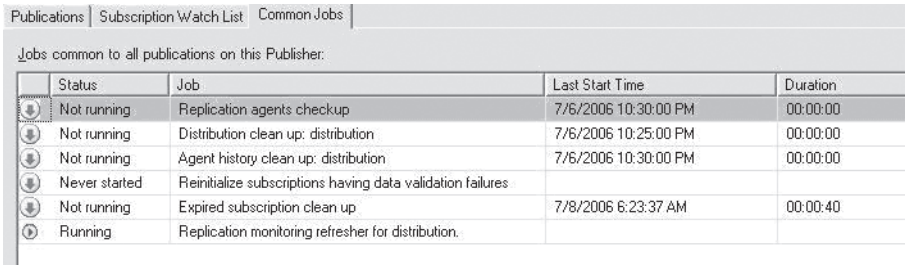


Рис. 3

Можно щелкнуть правой кнопкой мыши любое из этих заданий, чтобы просмотреть его свойства, запустить или остановить его. Управлять заданиями репликации (а также заданиями других типов) можно также с помощью монитора Job Activity, показанного на следующее рисунке. Наличие пользовательского интерфейса, ориентированного на основные задачи репликации, весьма удобно, если на вашем сервере сотни других задач.

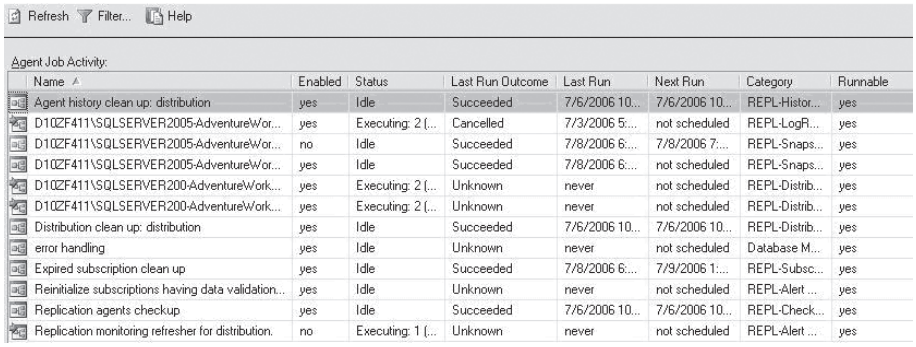


Рис. 4

Обратите внимание, что у монитора Job Activity есть механизм фильтрации, который помогает сократить число отображаемых заданий; например, на следующем рисунке показаны только задания, в названии которых содержится слово «*distribution*».

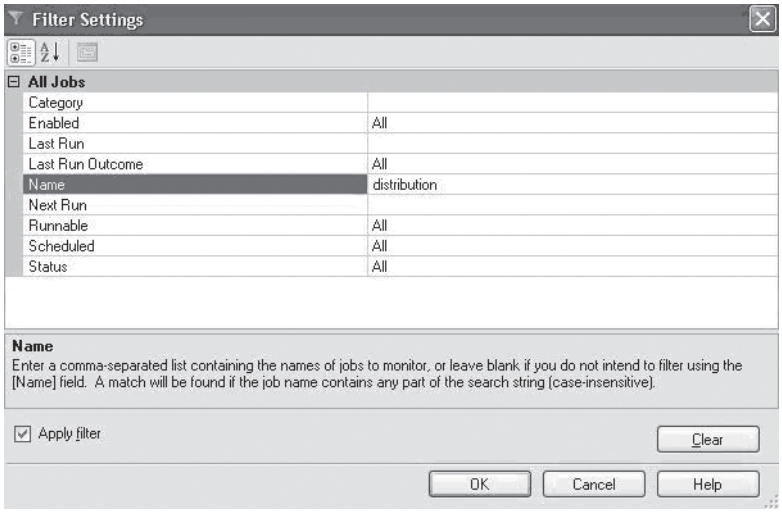


Рис. 5

Устранение ошибок репликации

Пока что вы видели только краткий статус публикаций и общие задания репликации, но что если в какой-то из ваших публикаций возникла проблема? Как просмотреть отчет о действиях агента мгновенных снимков, агента чтения журнала и распределительного агента? Давайте внимательно посмотрим на вкладку Subscription Watch List, которая доступна при подключении к любому издателю. Если щелкнуть подписку на этой вкладке правой кнопкой мыши, вы увидите все, что искали.

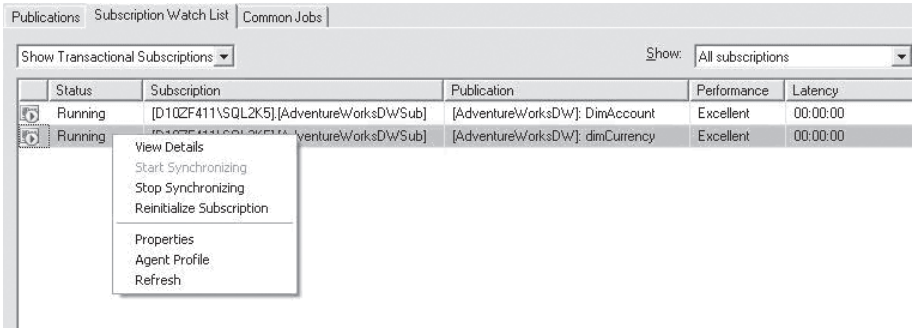


Рис. 6

Щелкнув правой кнопкой распределительного агента, можно просмотреть отчет о его действиях, запустить или остановить синхронизацию подписки, повторно инициализировать или обновить ее, просмотреть детали профиля распределительного агента или свойства распределительного задания.

В отличие от предыдущих версий, в SQL Server 2005 для просмотра деталей подписки в Replication Monitor существует отдельный экран с несколькими вкладками, как показано ниже.

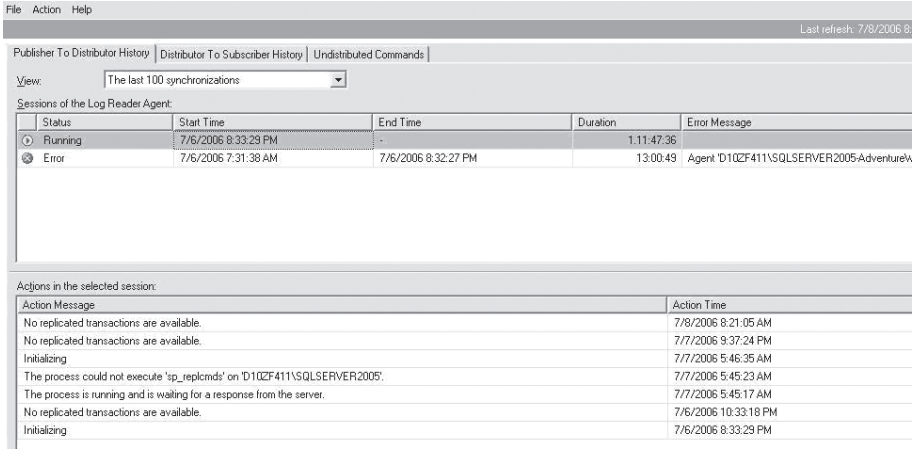


Рис. 7

На этом экране есть три вкладки; первая называется Publisher to Distributor History и показывает информацию о действиях агента чтения журнала, а вторая – о действиях распределительного агента, как показано здесь:

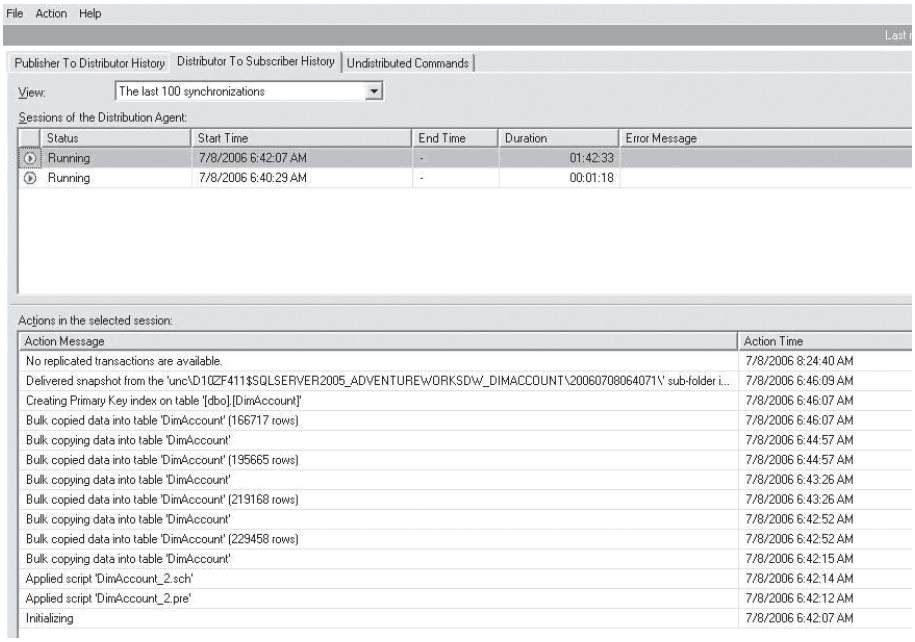


Рис. 8

Примечание. Replication Monitor опрашивает серверы каждые несколько секунд, чтобы продемонстрировать вам актуальные данные. Если вы хотите обновлять любой из экранов чаще, нажмите F5.

Поначалу вам, возможно, будет непривычно пользоваться двумя отдельными вкладками для просмотра статуса распределительного агента и агента чтения журнала, потому что в предыдущих версиях сведения по обоим агентам были доступны с одного экрана. Впрочем, даже тогда приходилось открывать новое окно, чтобы просмотреть историю по каждому из агентов; разница в том, что в SQL Server 2000 приходилось закрывать окно с историей распределительного агента, чтобы просмотреть сведения о действиях агента чтения журнала. Теперь история по каждому из агентов доступна через одно диалоговое окно на разных его вкладках.

На обоих вкладках число отображаемых синхронизаций можно ограничить, доступны следующие варианты:

- Все синхронизации
- Синхронизации, в ходе которых произошли ошибки
- Синхронизации за последние 24 часа
- Синхронизации за последние 2 дня
- Синхронизации за последние 7 дней
- Последние 100 синхронизаций

Более того, через меню Action можно стартовать/останавливать синхронизацию, просматривать свойства подписки, повторно инициализировать подписку, настраивать ручное или автоматическое обновление экрана и просматривать свойства агентов репликации.

Третья вкладка на этом экране представляет собой одно из лучших нововведений в области мониторинга – она показывает общее количество команд в распределительной базе данных, которые ждут доставки подписчикам, и примерное время доставки этих команд.

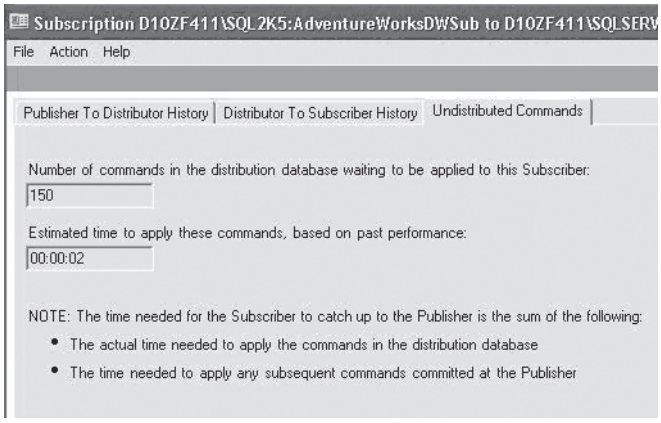


Рис. 9

В средствах мониторинга репликации предыдущих версий SQL Server такая функциональность отсутствовала. Чтобы определить число ожидающих доставки команд, можно было послать запрос к системным таблицам в распределительных базах данных, но более-менее точно оценить время их доставки было невозможно.

Если вы хотите изучить историю действий агента мгновенных снимков или чтения журнала, щелкните нужную публикацию в левой панели Replication Monitor и перейдите на вкладку Warnings and Agents. В нижней панели на этой вкладке можно выбрать агента и просмотреть его историю, щелкнув его правой кнопкой мыши и выбрав в контекстном меню View Details, как показано ниже.

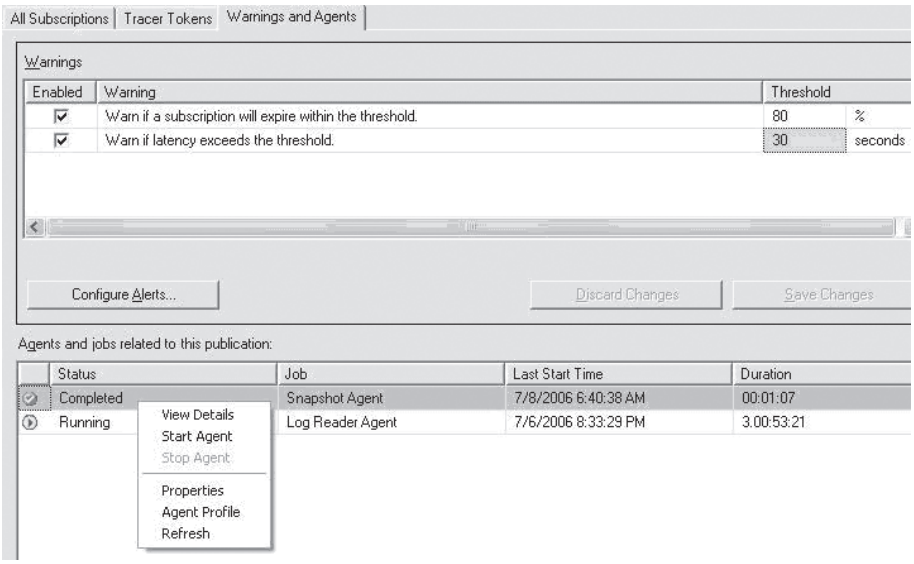


Рис. 10

Этот пользовательский интерфейс напоминает экраны для администрирования репликации в предыдущих версиях SQL Server. В SSMS (SQL Server Management Studio) есть другой удобный способ проверить статус каждого агента репликации и, если нужно, запустить его. Щелкнув правой кнопкой мыши публикацию в локальной папке publications, вы увидите следующие варианты: View Snapshot Agent Status и View Log Reader Agent Status. А если щелкнуть правой кнопкой любую подписку, можно будет проверить статус распределительного агента, выбрав пункт View Synchronization Status. Например, на следующем снимке экрана показан статус агента чтения журнала.

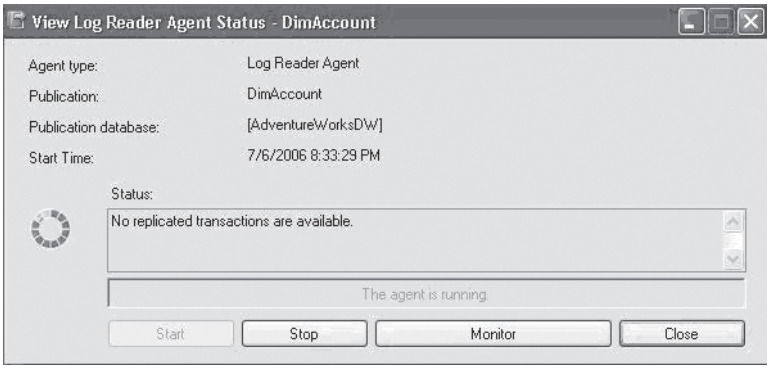


Рис. 11

Щелчок по кнопке Monitor на этом экране запускает Replication Monitor. Статус агентов моментальных снимков и чтения журнала отображается точно в таком же окне.

Теперь давайте сознательно нарушим процесс репликации, чтобы посмотреть, как Replication Monitor помогает в решении проблем. Сначала я удалю строку из базы данных на подписчике, а потом ту же самую строку на издателе.

```
DELETE dimCurrency
WHERE CurrencyKey = 1
```

Запущенное на издателе выражение DELETE будет реплицировано, но на подписчике не будет найдена нужная строка. Replication Monitor тут же сообщит об ошибке, как показано ниже.

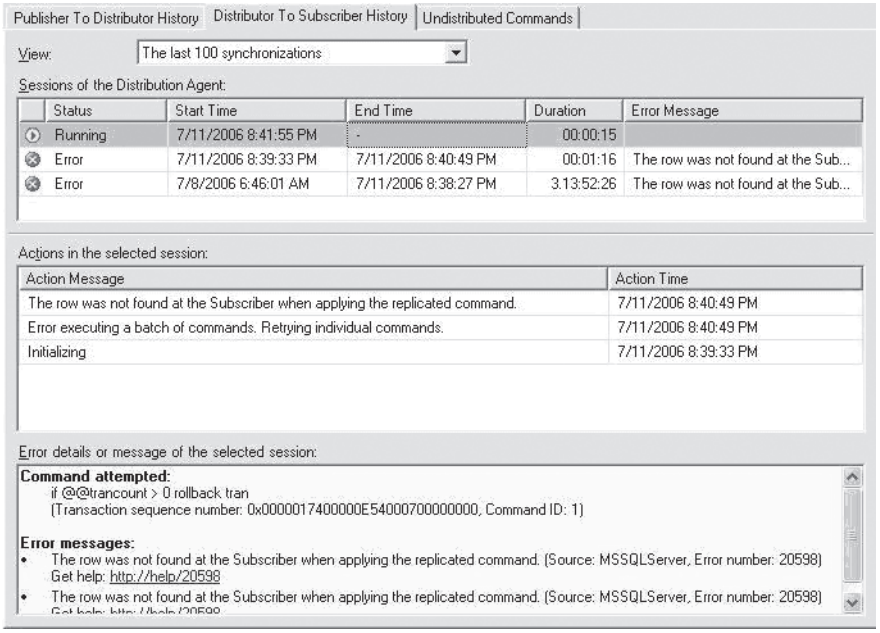


Рис. 12

Пока все правильно. Но как решить проблему? Инструмент сообщает нам, что на подписчике не найдена нужная строка, но не говорит, какая команда вызвала ошибку. К сожалению, в SQL Server 2000 Replication Monitor это было реализовано лучше. Здесь же для обнаружения проблемной команды нам придется выполнить дополнительную операцию, а именно скопировать предоставленный Replication Monitor порядковый номер транзакции и воспользоваться системной процедурой sp_browsereplcmds (в распределительной базе данных), чтобы отследить выражение, которое привело к ошибке репликации:

```
EXEC sp_browsereplcmds '0x0000017400000E54000700000000'
```

Результаты:

```
Xact_seqno                  originator_srvname  Originator_db  article_id type  partial_command
0x0000017400000E540007  NULL                NULL           10         30    0

hashkey originator_publication_id  originator_db_version      originator_lsn
1       NULL                       NULL                      0x00000000000000000000

Command                                  command_id
{CALL [sp_MSdel_dboDimCurrency] (1)}     1
```

Теперь мы знаем, что репликацию прервало выполнение sp_MSdel_dboDimCurrency с параметром 1. Есть два простых метода исправления этой ошибки:

- Добавить на подписчике запись с currency_id = 1. Поскольку в процессе репликации эта запись будет удалена, ее содержимое не имеет значения; мы добавляем ее лишь для того, чтобы репликация прошла нормально.
- Сконфигурировать агента репликации так, чтобы он пропустил ошибку, вызванную отсутствием строки: ошибку номер 20598. Для этого можно изменить профиль распределительного агента, открыть который можно, щелкнув агента правой кнопкой мыши и выбрав Agent Profile. При этом отобразятся профили, созданные по умолчанию, как показано на следующем рисунке.

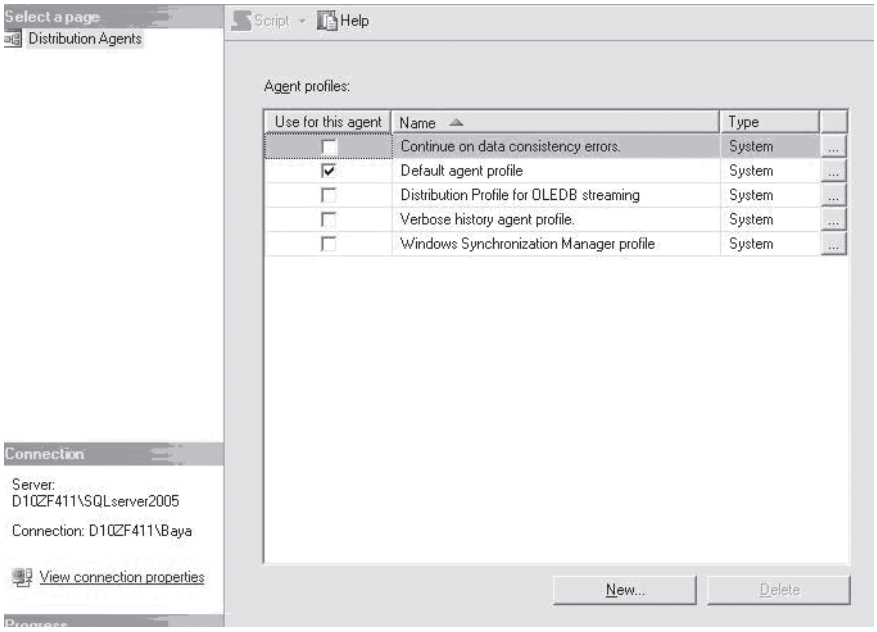
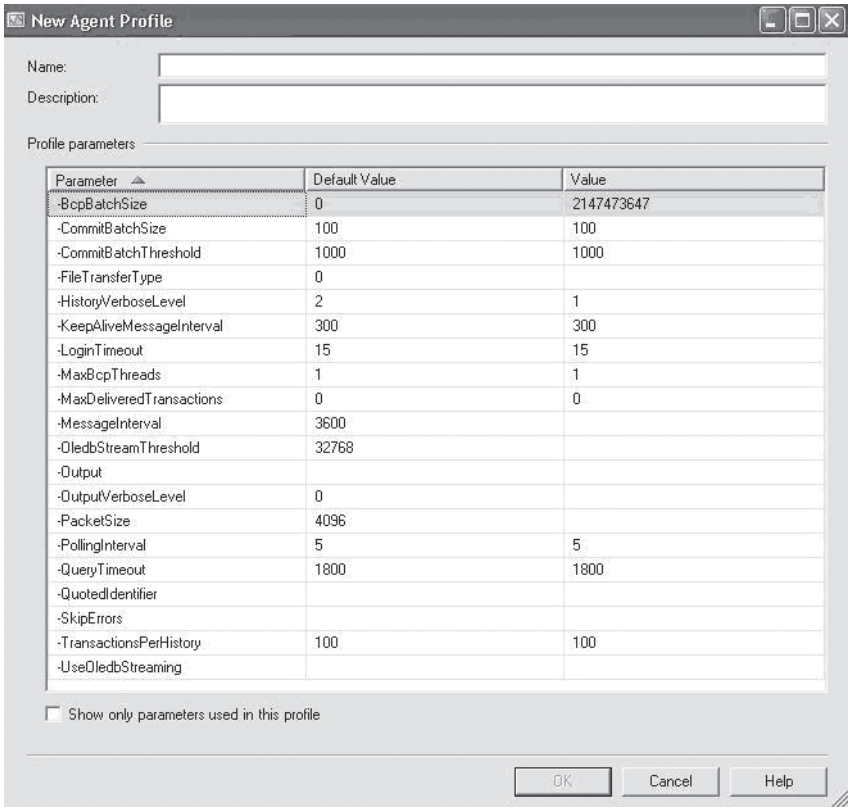


Рис. 13

Щелкнув кнопку с многоточием напротив любого профиля, можно просмотреть настройки этого профиля. Однако чтобы изменить настройки по умолчанию, нужно создать новый профиль. Экран для конфигурирования новый параметров распределительного агента открывается по кнопке New (он приведен ниже).



Примечание. После смены профиля распределительного агента, его нужно перезапустить, чтобы изменения вступили в силу.

Проверка подписки

Replication Monitor позволяет проверять подписку на предмет соответствия данных на подписчике и издателе. При этом можно проверять только число строк, число строк и двоичную контрольную сумму или число строк и контрольную сумму. Чтобы выбрать подписки для проверки, щелкните правой кнопкой мыши публикацию в левой панели Replication Monitor и выберите «Validate Subscriptions...». Откроется следующий диалог:

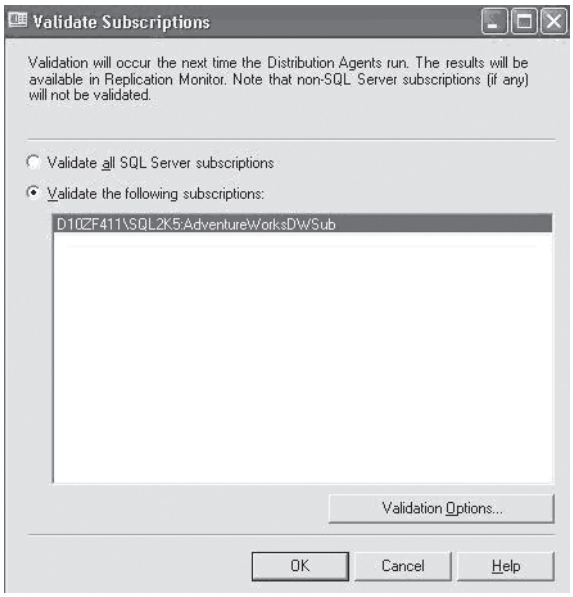


Рис. 15

Щелкнув Validation Options, можно сообщить SQL Server, какой алгоритм использовать для проверки данной подписки, как показано ниже.

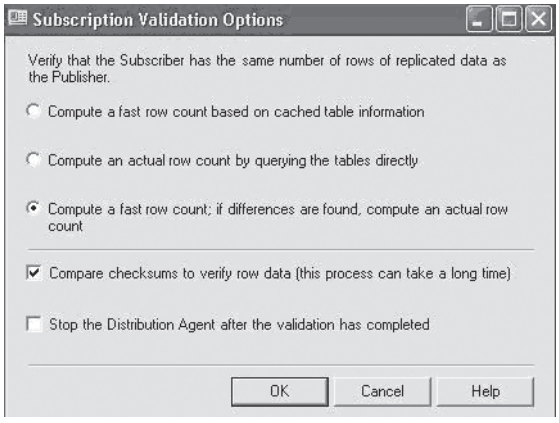


Рис. 16

Экран предупреждает, что некоторые варианты проверки требуют длительного времени и создают дополнительную нагрузку на сервер, так что постарайтесь проверять подписки в периоды наименьшей активности пользователей. После того, как вы указали способ проверки, распределительный агент выведет для каждой опубликованной таблицы сообщение примерно следующего содержания:

Table 'DimAccount' passed rowcount (811218) and checksum validation. Checksum is not compared for any text or image columns.

Эту информацию можно просмотреть на вкладке Distributor To Subscriber History в разделе Action Message.

Если вы используете отложенное обновление подписчиков (с организацией очереди), SSMS позволяет просмотреть возникающие при репликации конфликты. Конфликт происходит, если одну и ту же строку изменяют и на издателе, и на подписчике. Обсуждение конфликтов репликации выходит за рамки этой статьи, но имейте в виду, что для разрешения любых конфликтов можно щелкнуть правой кнопкой мыши публикацию с отложенным обновлением и выбрать View Conflicts.

Мониторинг и решение проблем производительности репликации

Replication Monitor был значительно улучшен по сравнению с предыдущими версиями, и особенно это заметно в области мониторинга производительности репликации. Вы уже видели, что на вкладке Publications есть два столбца для отслеживания производительности: Current Average Performance и Current Worst Performance. В зависимости от порогового значения задержки при репликации в этих столбцах отображаются разные величины, которые соответствуют следующим показателям:

- Excellent: Задержка 0–34% от порогового значения. Например, если пороговое значение задержки 30 секунд, а транзакции доставляются за 3, то производительность превосходная.
- Good: Задержка 35–59 % от порогового значения.
- Fair: Задержка 60–84 % от порогового значения.
- Poor: Задержка 85–99 % от порогового значения.
- Critical: Задержка превышает пороговое значение.

Задержка при репликации – это промежуток времени, необходимый для фиксации транзакции в базе данных подписчика после ее фиксации в публикуемой базе данных. По умолчанию пороговое значение задержки при репликации составляет 30 секунд; чтобы просмотреть и изменить этот параметр нужно выбрать публикацию в левой панели и перейти на вкладку Warnings And Agents, как показано на следующем рисунке.

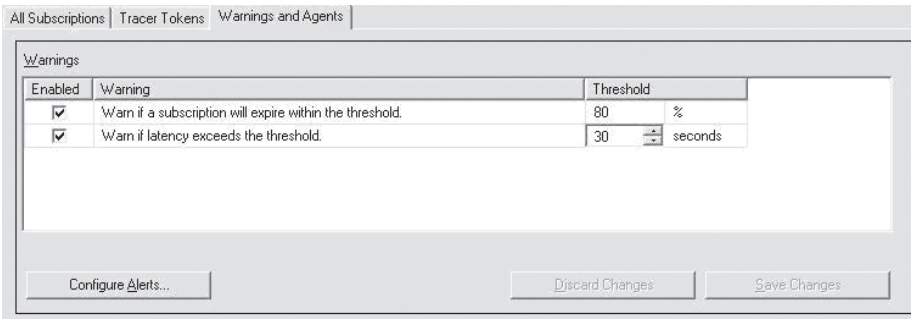


Рис. 17

Если в вашей среде задержка при репликации не играет заметной роли, можно отключить предупреждения, сняв флажок Enabled. При применении самого первого мгновенного снимка задержка более 30 секунд является нормой; если вы не хотите постоянно видеть показатель Critical, просто поднимите пороговое значение до разумной величины.

Replication Monitor позволяет настроить оповещения, тогда при выполнении некоторого связанного с производительностью условия администратор об этом узнает. Или можно запускать задание при выполнении такого условия. Список связанных с репликацией оповещений доступен через приведенное на следующем рисунке окно.

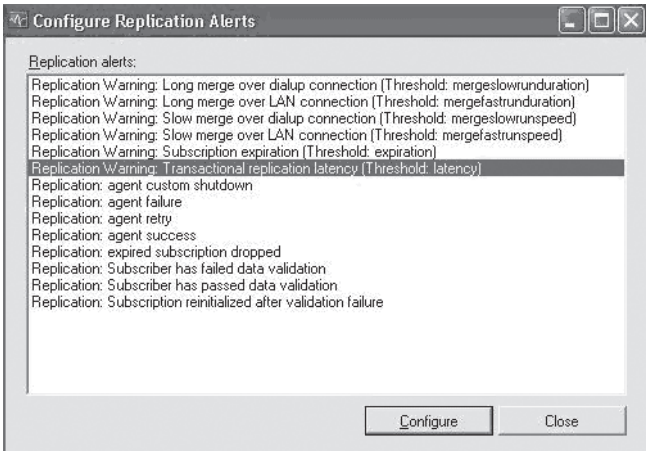


Рис. 18

По щелчку на кнопке Configure открывается диалоговое окно для настройки свойств оповещений. Оповещения, связанные с репликацией, настраиваются примерно так же, как все остальные оповещения SQL Server, так что я не буду подробно на них останавливаться.

Еще одно улучшение в области мониторинга производительности репликации реализовано через маркеры слежения. Если щелкнуть некото-

рую публикацию в левой панели Replication Monitor, в правой появится три вкладки. Первая просто демонстрирует все подписки на данную публикацию и позволяет получить более полную информацию по конкретной подписке. Вторая вкладка, Tracer Tokens, позволяет вставлять в журнал транзакций публикуемой базы данных небольшие участки данных (маркеры слежения), чтобы засечь время, за которое эти данные дойдут до распределительного сервера и подписчика. Как видно из следующего рисунка, у рассматриваемой подписки общая задержка составляет пять секунд (четыре из них ушло на доставку транзакции дистрибьютору, а последняя секунда – на ее доставку от дистрибьютора к подписчику).

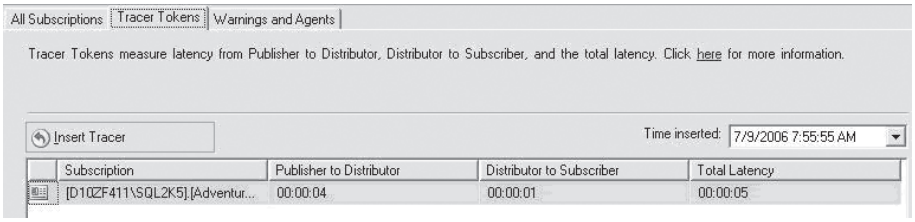


Рис. 19

Маркеры слежения – простой способ подсчета производительности публикации и отслеживания связанных с производительностью проблем. В этом примере я установил, что доставка данных от издателя к дистрибьютору занимает четыре секунды; если бы это значение было выше нормы, я бы проверил канал связи между этими серверами. Более того, возможно, я бы решил использовать удаленный распределительный сервер или запросную подписку вместо принудительной, чтобы снизить нагрузку на публикующий сервер. Если бы задержка при доставке транзакций от дистрибьютора к подписчику была неприемлемой, я бы проверил канал связи между ними, а также любую другую активность на подписчике, которая могла замедлить получение реплицированных команд.

Системные процедуры для мониторинга репликации

Существует несколько системных хранимых процедур для программного мониторинга репликации и выявления причин возникновения ошибок. В большинстве случаев вы, как и в более ранних версиях SQL Server, будете пользоваться процедурой sp_browsereplcmds. Хотя эту процедуру можно запускать без параметров, можно задать начальный и конечный порядковый номер транзакции, идентификатор команды, агента, источника транзакции, статьи, а также уровень совместимости. (Я уже приводила пример использования sp_browsereplcmds для просмотра реплицируемых команд в этой статье).

Процедуру sp_replmonitorhelppublisher можно применять для получении информации об издателях, которые используют в качестве дистрибьютора текущий сервер; ее можно запустить без параметров для получения сведений обо всех издателях или указать имя издателя, чтобы получить данные только о нем. Например:

```
EXEC sp_replmonitorhelppublisher
```

Результаты:

publisher	distribution_db	Status	warning	publication count	returnstamp
D10ZF411\SQLSERVER2005	Distribution	4	0	2	2006071219205930

Столбец status может принимать одно из следующих значений:

- 1 – запущена
- 2 – успешно завершилась
- 3 – выполняется
- 4 – простой
- 5 – повторная попытка
- 6 – закончилась неудачей

Столбец Warning может принимать несколько различных значений в зависимости от статуса подписки.

Чтобы осведомиться о «здоровье» данной публикации и ассоциированных с ней подписок, можно воспользоваться двумя очень похожими процедурами (sp_replmonitorhelppublication и sp_replmonitorhelpsubscription), обе они являются новшеством SQL Server 2005. Детальное описание и примеры использования этих процедур вы можете найти в онлайновой документации.

Заключение

В этой статье был представлен Replication Monitor – прекрасный инструмент для мониторинга и решения проблем в SQL Server 2005. Средства мониторинга репликации в SQL Server 2005 значительно улучшились по сравнению с предыдущими версиями. Особенно удобно иметь возможность мониторинга производительности репликации в реальном времени, а также иметь представление о времени доставки реплицируемых транзакций, находящихся в распределительной базе данных в данный момент. Администраторы баз данных могут настраивать оповещения, чтобы своевременно узнавать о проблемах в области репликации. Хотя Replication Monitor и SSMS в большинстве случаев должно быть достаточно, вам стоит познакомиться с несколькими системными хранимыми процедурами, которые при необходимости помогут вам получить более подробную информацию.

Агенты репликации транзакций в SQL Server 2005

По материалам статьи Байя Павлиашвили (Baya Pavliashvili) «SQL Server 2005 Transactional Replication Agents»

Перевод Александра Сергеевича Волока

Агенты репликации в SQL Server 2005 уже изначально настроены оптимально. Но иногда может потребоваться изменять поведение агентов по умолчанию, редактируя их параметры. Посредством графического интерфейса SQL Server 2005 предоставляет доступ к некоторым параметрам, которые в предыдущих выпусках были доступны только из командной строки или через шаги заданий агентов. Байа Павлиашвили предлагает изучить эти новые опции, доступные для всех агентов репликации, и то, как тонко настраивать эти опции для ваших приложений.

В некоторых из моих предыдущих статей обсуждалась настройка, мониторинг, решение проблем и обслуживание транзакционной репликации в SQL Server 2005. В этой статье рассматривается внутреннее функционирование агентов репликации: безопасность, свойства и профили. Также я представлю несколько системных процедур для программного управления агентами репликации.

Если вам необходимо освежить в памяти то, какие существуют агенты репликации, и какие у них имеются параметры в предыдущих версиях SQL Server, не забудьте ознакомиться с моей предыдущей статьей «SQL Server Transaction Replication».

В этой статье будут рассмотрены улучшения в SQL Server 2005, а также то, какую пользу вы можете извлечь из этих улучшений.

Безопасность агентов

Вспомните, что агенты репликации предоставлены как задания агента SQL Server, и на самом деле, это обычные исполняемые файлы, так что они могут запускаться из командной строки.

Репликация транзакций использует агенты моментальных снимков, агенты чтения журнала и агенты распространителя; если разрешены подписчики с отложенным обновлением, также задействуется агент чтения очереди.

В предыдущих версиях SQL Server агенты репликации обычно выполнялись под учетной записью агента SQL Server. В обычных сценари-

ях, репликация перемещает транзакции между компьютерами, а не только между базами данных на одном компьютере. В таких случаях агент SQL Server должен выполняться в контексте доменной учетной записи (не локальной). В качестве альтернативы при соединении с серверами-издателями, распространителями и подписчиками может использоваться аутентификация SQL Server.

В SQL Server 2005 безопасность агентов репликации работает аналогично; вы все еще можете использовать олицетворение (impersonate) агента SQL Server или же указать использование учетной записи SQL Server для соединения с серверами, участвующими в репликации. К тому же вы можете указать учетную запись Windows, отличную от той, под которой запущен агент SQL Server. Почему это важно? Вообще нередко можно увидеть среды, в которых, по разным причинам, агент SQL Server выполняется под учетной записью администратора домена; агенты репликации не нуждаются в правах администратора домена при выполнении своих действий. Новая опция позволит вам более серьезно отнестись к безопасности, и запускать агент под учетной записью с правами, которых будет достаточно для осуществления репликации.

К примеру, агенту моментальных снимков необходимы права доступа для подключения к каталогу со снимками, а также для чтения и записи данных в этом каталоге. Для этого можно создать учетную запись Windows, которая является локальным администратором на издателе, но не имеет никаких прав доступа на подписчике, и при выполнении агента использовать эту учетную запись. Аналогично, учетная запись агента распространителя для подписки по запросу должна иметь право на соединение с распространителем, но вовсе не нуждается в правах для доступа к издателю. Вы можете указать учетные записи для каждого агента репликации, используя диалоговое окно Agent Security.

Примечание. Диалоговое окно предупреждает, что выполнение агентов репликации под учетной записью агента SQL Server не самый оптимальный и рекомендованный метод.

Если вы изменили свое решение и хотите олицетворять агент SQL Server вместо использования учетной записи Windows, вы можете изменить свой выбор в свойствах шага задания. Откройте шаг задания в котором запускается агент, и выберите SQL Server Agent Service Account в выпадающем списке Run As.

Профили агентов

В профилях агентов репликации задаются значения различных параметров, которые можно использовать для каждого запуска агента. Все агенты имеют многочисленные опции, которые можно использовать для тонкой настройки. В этом разделе будут рассмотрены только те опции,

которые появились в SQL Server 2005 или были изменены по сравнению с предыдущей версией.

Для просмотра параметров в текущем профиле агента репликации, нажмите правую кнопку на агенте и выберите Agent Profile, нажмите на кнопку с многоточием около названия профиля. Параметры встроенных профилей изменять невозможно. Вместо этого, нажмите New для создания нового профиля; SQL Server предложит выбрать встроенный профиль, на основании значений параметров которого будет создан новый профиль. Например, если создается новый профиль для агента чтения журнала, для его инициализации можно выбрать профиль агента по умолчанию или профиль агента с подробным ведением истории. Обратите внимание на флажок внизу диалогового окна настройки параметров профиля: Show Only Parameters Used In This Profile. Эта опция чрезвычайно удобна, если необходимо изменить только несколько параметров без необходимости просматривать множество других. Также, просто нажимая на названия столбцов, можно упорядочивать параметры по названию параметра, значению по умолчанию или текущему значению.

Примечание. В SQL Server 2005 появилась возможность задать некоторые параметры (к примеру, параметр OUTPUT) в профиле агента, которые в предыдущих версиях невозможно было задать с помощью профиля. Но посредством профилей все еще невозможно изменять все параметры агентов репликации. Некоторые параметры могут быть изменены лишь правкой шагов заданий агента (об этом будет рассказано в следующем разделе).

В следующей таблице приведены новые параметры агентов репликации SQL Server 2005, и параметры, измененные по сравнению с предыдущими версиями.

Агенты	Параметр	Описание/ примечания	Появился в SQL Server 2005	Значение по умолчанию
Моментальных снимков, чтения журналов, распространитель, чтения очереди	Encryption Level	Репликация не шифрует данные, пересылаемые от издателя к подписчикам; однако в SQL Server 2005 может использоваться SSL для шифрования данных, передаваемых по сети. Данная статья не охватывает подробности настройки SSL, но для этой цели можно воспользоваться вкладкой Network Configuration в SQL Server Configuration Manager.	Да	0

Агенты	Параметр	Описание/ примечания	Появился в SQL Server 2005	Значение по умолчанию
		Параметр Encryption Level означает используемый агентом уровень шифрования и может содержать следующие значения: 0: SSL не используется. 1: SSL используется, но агент не проверяет сертификаты, подписанные доверенными запрашивающими сторонами. 2: SSL используется, и сертификаты проверяются.		
Распространитель	OLEDB Stream Threshold	Этот параметр важен, только если включена опция OLEDB Streaming для больших двоичных объектов.	Да	32768
Распространитель	Use In Proc Loader	Принуждает агентов использовать команду Bulk Insert для применения моментальных снимков. Эта опция может повысить производительность доставки моментальных снимков. Не забывайте, однако, что этот параметр несовместим с типом данных XML.	Да	0
Распространитель	Use OLEDB Streaming	Задействуется связывание больших бинарных объектов данных в потоки. Значение 1 значит, что OLEDB Streaming включено; 0 - эта опция отключена.	Да	0
Моментальных снимков	70 Subscribers	Этот параметр обязателен, если какие-либо подписчики используют SQL Server 7.0.	Да	0
Моментальных снимков	Max Network Optimization	Этот параметр важен только для репликации слиянием и используется для уменьшения вероятности выполнения излишних команд на подписчике.	Да	0

Агенты	Параметр	Описание/ примечания	Появился в SQL Server 2005	Значение по умолчанию
Моментальных снимков, чтения журналов, чтения очереди	Publisher Failover Partner	Если используется зеркальное отображение базы данных, этот параметр указывает для издателя имя сервера или имя экземпляра партнера зеркального отображения.	Да	None
Моментальных снимков	Start Queue Timeout	Максимальное количество секунд ожидания агента при выполнении параллельных динамических моментальных снимков. Этот параметр важен только для репликации слиянием.	Нет	1800
Моментальных снимков, чтения журналов, распространитель, чтения очереди	Query Timeout	Количество секунд до отказа агента по причине превышения лимита времени для запроса. Значение по умолчанию увеличено с 300 до 1800 секунд.	Нет	1800
Моментальных снимков, чтения журналов, распространитель, чтения очереди	Output	Хотя этот параметр был доступен и в прежних версиях, его нельзя было указать, используя пользовательский графический интерфейс профилей агентов.	Нет	None
Моментальных снимков, чтения журналов, распространитель, чтения очереди	Output Verbose Level	Хотя этот параметр был доступен и в прежних версиях, его нельзя было указать, используя пользовательский графический интерфейс профилей агентов.	Нет	None
Чтения журналов	Read Batch Threshold	Количество команд, которые должны быть прочитаны из журнала транзакций в одном пакете перед отправкой этих команд подписчикам. Этот параметр не является новым, но значение по умолчанию изменено на 0 (принуждает SQL Server читать все реплицированные команды в журнале транзакций), а не 100.	Нет	0

Агенты	Параметр	Описание/ примечания	Появился в SQL Server 2005	Значение по умолчанию
Чтения журналов	Recover from Data Errors	Уведомляет SQL Server продолжать репликацию от издателя Oracle, даже если произошла ошибка на уровне данных столбцов. Если этот параметр имеет значение 0, такие ошибки вызовут остановку агента чтения журналов. Если значение равно 1, для столбцов, вызвавших ошибку, будет реплицироваться значение NULL.	Да	0
Чтения журналов, чтения очереди	Polling Interval	Определяет частоту опроса журнала транзакций издателя. В предыдущих версиях значение по умолчанию было равно 10 секундам.	N	5
Моментальных снимков	HRBCP Dynamic Blocks	Специфично для издателей Oracle, этой опцией контролируется, могут ли блоки данных Bulk Copy Program (BCP) увеличиваться динамически.	Да	None
Моментальных снимков	HRBCP BlockSize	Специфично для издателей Oracle, этой опцией контролируется размер блоков BCP (в килобайтах).	Да	64
Моментальных снимков	HRBCP Blocks	Специфично для издателей Oracle, этот параметр управляет количеством ожидающих в очереди блоков BCP между потоками чтения и записи.	Да	50
Распространитель	Subscription Streams	Управляет количеством процессов, которые могут подключаться к подписчику и параллельно доставлять реплицированные команды. Максимальное значение равно 64. Поддерживается только подписчиками SQL Server 2005. Для предыдущих версий этот параметр игнорируется.	Да	0

Агенты	Параметр	Описание/ примечания	Появился в SQL Server 2005	Значение по умолчанию
Распространитель	Use DTS	Этот параметр необходим, если публикация позволяет выполнять трансформацию данных перед отправкой их подписчикам.	Да	None
Моментальных снимков	Publisher Deadlock Priority	Опционально назначает при взаимной блокировке приоритет соединению, создающему моментальный снимок на издателе. По умолчанию приоритет не устанавливается. Значение 1 предоставляет повышенный приоритет по сравнению с другими приложения. Значение (-1) назначает пониженный приоритет.	Да	0

Свойства агентов

В SQL Server 2005, как и в предыдущих версиях, каждый агент репликации реализован в виде задания с тремя шагами. В первом шаге протоколируется стартовое сообщение агента, во втором шаге запускается агент с набором параметров, и в последнем шаге выполняется системная процедура sp_MSdetect_nonlogged_shutdown, которая протоколирует историю выполнения агента. Если нажать правую кнопку на агенте чтения журналов или на агенте моментальных снимков любой публикации в Мониторе Репликаций и выбрать пункт Agent Properties, вы увидите свойства задания, которое запускает агента. Параметры агентов можно задать во втором шаге задания.

Любопытно, что если в контекстном меню агента распространителя выбрать свойства, это вызовет свойства подписчика вместо отображения шагов задания. Мне не известна причина такой противоречивости пользовательского интерфейса Монитора Репликаций. К счастью, попрежнему можно исследовать свойства агента распространителя, перейдя в агент SQL Server, и далее в задания, где необходимо нажать правую кнопку на задании распространителя и выбрать его свойства.

Все параметры, которые поддерживаются агентами, можно указать в шагах их задания. Если какой-либо из параметров не указан явно, агентом будет принято значение по умолчанию для этого параметра.

Также возможно запускать агентов репликации с помощью командной строки. Если это требуется сделать, перейдите в директорию, в ко-

торой установлен SQL Server, и запустите утилиты snapshot.exe или distrib.exe, которые можно найти в каталоге 90\Com.

Управление агентами репликации с помощью хранимых процедур

В SQL Server 2005 включено несколько системных процедур, с помощью которых возможно программно управлять профилями и параметрами агентов. Хотя, вероятно, вы будете управлять профилями агентов, используя SSMS, осведомленность об этих хранимых процедурах станет полезной, если под вашим управлением будет большое количество серверов и возникнет необходимость автоматизировать административные задачи. Все хранимые процедуры, которые будут рассмотрены в этом разделе, необходимо запускать в контексте базы данных распространителя.

Процедура sp_help_agent_profile возвращает идентификаторы профилей для агентов заданного типа репликации (моментальных снимков, распространителя, чтения журналов, чтения очереди, слияния). К примеру, результатом запуска процедуры будет список всех профилей агента распространителя, определенных в данном экземпляре SQL Server:

Результат (в сокращенном виде):

Profile ID	Profile Name	Description	def_profile
4	Default agent profile	Null.	1
5	Verbose history agent profile.	Agent profile for detailed history logging.	0
10	Windows Synchronization Manager profile	Profile used by the Windows Synchronization Manager.	0
14	Continue on data consistency errors	Agent profile for skipping data consistency errors. It can be used only by SQL Server Subscribers.	0
16	Distribution Profile for OLEDB streaming	Distribution agent profile enabled for the processing LOB data using OLEDB streaming.	0
17	User profile	Null.	0

Процедура sp_help_agent_parameter извлекает параметры для указанного профиля; в качестве необходимого параметра для этой процедуры можно использовать идентификатор, полученный посредством процедуры sp_help_agent_profile. К примеру, результатом выполнения команды:

```
EXECUTE sp_help_agent_parameter  @profile_id = 10
```

будет вывод параметров для профиля Windows Synchronization Manager:

Profile ID	Parameter Name	Value
10	BcpBatchSize	2147473647
10	CommitBatchSize	100
10	CommitBatchThreshold	1000
10	HistoryVerboseLevel	1
10	KeepAliveMessageInterval	300
10	LoginTimeout	15
10	MaxBcpThreads	1
10	MaxDeliveredTransactions	0
10	PollingInterval	5
10	QueryTimeout	1800
10	SkipErrors	**Нет значения!**
10	TransactionsPerHistory	100

Примечание. Эта процедура возвращает не все доступные параметры, а только те, которые были использованы в указанном профиле. Если необходимо изменить профиль, включая или исключая некие параметры, необходимо воспользоваться системными процедурами sp_add_agent_parameter или, соответственно, sp_drop_agent_parameter.

Резюме

В этой статье показаны некоторые сложные моменты агентов репликации в SQL Server 2005. Также представлено несколько системных хранимых процедур, которые могут помочь автоматизировать некоторые рутинные задачи, связанные с агентами репликации.

Общее же поведение агентов репликации не изменилось, но в новой версии представлены некоторые приятные улучшения, как в безопасности, так и в параметрах профилей агентов.

Репликация хорошо функционирует изначально, но знание внутренней функциональности агентов поможет вам решать любые проблемы, с которыми вы можете столкнуться.

Настройка одноранговой репликации транзакций

По материалам статьи Пауля Ибисона (Paul Ibison)
«Setting up Peer-to-Peer Transactional Replication»
Перевод Даутова Ильдара Ахметовича

Вступление

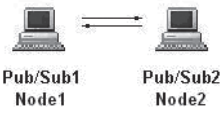


Рис. 1

Несмотря на наличие диаграмм и поясняющих материалов, присутствующих в Интернет, одноранговая репликация – это не топология «каждый может изменять все на каждом узле», и не готовая топология восстановления после сбоев! Это – действительно неиерархическая установка, какой мы никогда не имели прежде: все узлы являются равными, каждый узел – одновременно и издатель, и подписчик, никакой узел не является единственно ответственным. Данное решение может использоваться и для распределения рабочей нагрузки между серверами, и для восстановления в случае сбоя. Но есть несколько предостережений, о которых нужно знать и которые описаны ниже. Эта статья – не учебник по одноранговыи системам, и мы в основном сфокусируем свое внимание на простейших способах установки – вручную и с помощью мастера.

Ручная установка

Публикация создается как обычная транзакционная публикация, и это может быть сделано с использованием графического интерфейса. Обратите внимание, что в типах репликации нет никакого упоминания «peer-to-peer»:

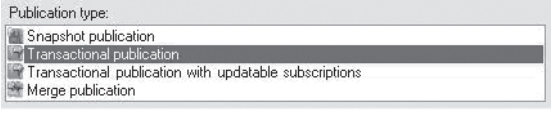


Рис. 2

Как тип репликации, опция «peer-to-peer» была удалена после первого релиза Yukon, но появилась в последующих версиях как отдельный пункт контекстного меню настройки существующей публикации. Для лучшего последующего понимания работы мастера, сначала мы покажем, как установить эту опцию вручную. Чтобы публикация была одноранговой после создания, необходимо в свойствах публикации на странице «Subscription» установить две опции: «Allow Initialization From Backup files» и «Allow peer-to-peer subscriptions».

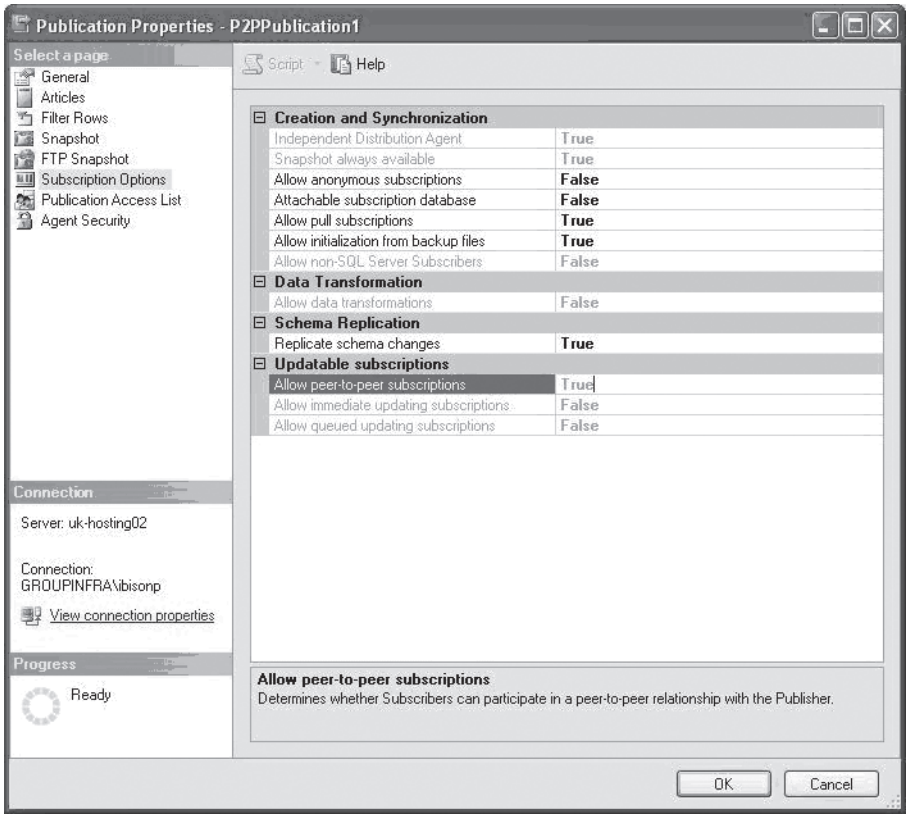


Рис. 3

Затем мы создаем копию базы данных на узле Node1 и восстанавливаем эту копию на узле Node2. Заметьте, что вы не можете создавать одноранговую репликацию на одном экземпляре сервера, так как требуются одинаковые имена публикации на каждом издателе и дубликаты публикаций не допускаются. Установка на одном сервере имеет смысл лишь для тестовых целей, и в этом случае вам потребуется установить два экземпляра SQL Server.

Для того, чтобы создать подписку на узле Node1 для узла Node2, воспользуйтесь мастером подписки, но не выполняйте действия до конца – просто создайте скрипт. Далее, отредактируйте вызов хранимой процедуры sp_subscription в скрипте, установив параметр @sync_type = «replication support only». Это значение параметра отменяет потребность в полной инициализации подписчика. Обратите внимание – необходимо гарантировать, что никакие изменения данных не были сделаны в базе данных публикации с тех пор как была создана копия. Запустите скрипт с новым значением параметра.

Теперь, когда у нас установлена репликация из узла Node1 в узел Node2, мы создаем обратные публикацию и подписку: публикация на Node2, подписка на Node1. Это может быть сделано либо посредством создания скрипта ранее созданной установки на узле Node1 и последующим его редактированием, или ручным повтором действий на узле Node2. Для публикаций, включающих много статей, я всегда пользуюсь скриптами. Не забудьте – если на одном из узлов используется SQL Server Developer Edition, то на этом этапе необходимо будет включить опцию «enable remote connections».

Теперь у нас есть готовая установка. Правда, не совсем. Мы должны учесть еще три момента.

(1) **Identities.** Если мы используем столбцы identity, необходимо гарантировать отсутствие перекрытия значений в разных узлах. В противном случае, при синхронизации возникнут конфликты по первичным ключам при добавлении записи в каждом узле.

(2) **Конфликты.** Нет разрешения конфликтов, и это большое отличие по сравнению с репликацией слиянием. Как избегать конфликтов? В основном – секционировать данные. Для Identity можно задать разные диапазоны, для других первичных ключей использовать идентификаторы, характеризующие месторасположение узла.

(3) **Избыточные данные.** На подписчике была восстановлена полная база данных. Эта копия содержит все необходимые статьи, а также другие объекты – таблицы, процедуры, функции и так далее. Лучше удалить лишние объекты, они только увеличивают нагрузку на администратора при поддержке системы.

Использование мастера

Как только мы выбрали опции «Allow Initialization From Backup files» и «Allow peer-to-peer subscriptions» в свойствах публикации, в контекстном меню публикации становится доступна новая опция «Configure Peer-To-Peer topology...»:

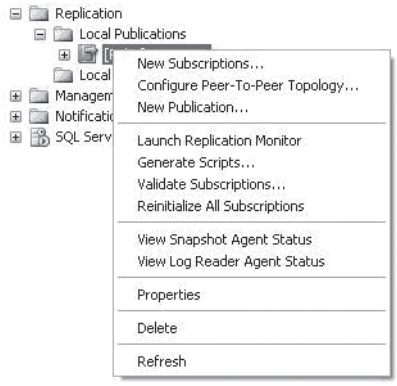


Рис. 4

По большому счету, мастер выполняет то, что мы описали ранее, но есть интересная альтернативная опция для определения того, как на подписчике восстановлена база данных:

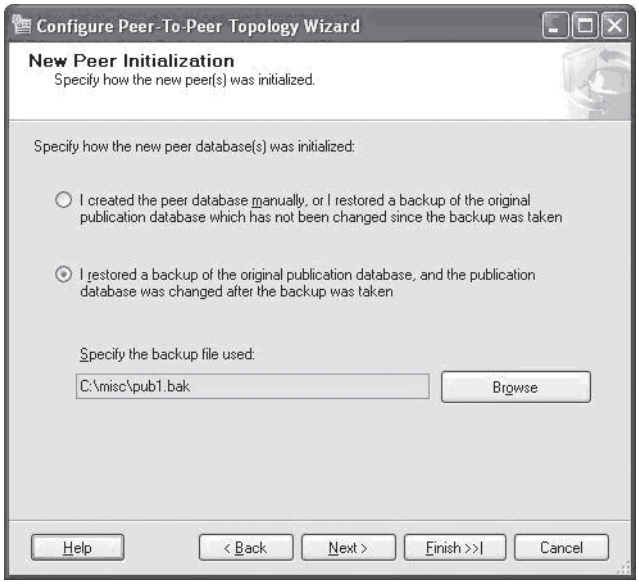


Рис. 5

Вы можете выбрать первую опцию, если у вас есть идентичная база данных на подписчике. Этот вариант подходит, если вы создали схему для опубликованных объектов вручную или восстановили копию, и никакие изменения данных с тех пор не были сделаны в первичной базе данных публикации. В скрипте эта опция соответствует значению «replication support only» для свойства подписки @sync_type.

Вторая опция используется, если после копирования данных были сделаны изменения в первичной базе данных публикации. Репликация должна теперь доставить те изменения первичной базы данных публикации, которые не были включены в копию. Эта опция соответствует значению «initialize with backup» для свойства подписки @sync_type.

Вы можете задаться вопросом – для чего нужна кнопка обзора (browse) на экране? Она доступна для второй опции и используется для нахождения файла резервной копии и определения из него регистрационного номера транзакции в журнале (log sequence number, LSN). Этот LSN сравнивается с записями в таблице MSRepl_Commands для передачи измененных записей.

Выводы

Настройка одноранговой репликации транзакций сравнительно проста. Мы можем использовать комбинацию SQL Server Management Studio, скриптов и мастеров для создания установки, и впоследствии выполнять лишь небольшие изменения, чтобы иметь рабочую систему.

Одноранговая репликация транзакций в Microsoft SQL Server 2005

По материалам статьи Диниш Асанка (Dinesh Asanka)
«Peer-to-Peer Transactional Replication in SQL Server 2005»

Перевод Алексея Николаевича Ковалева

Репликация это технология, позволяющая синхронизировать данные между несколькими базами данных. Как правило, репликация используется при необходимости разделить нагрузку для улучшения производительности серверов баз данных. Помимо этого, репликация помогает повысить доступность баз данных.

SQL Server 2000 предоставляет три типа репликации:

- Репликация моментальных снимков (snapshot replication)
- Репликация транзакций (transactional replication)
- Репликация слиянием (merge replication)

В SQL Server 2005 к вышеперечисленным методам репликации добавился новый:

- Одноранговая репликация транзакций (Peer-to-peer transactional replication)

Данная статья посвящена описанию нового метода **Одноранговой репликации транзакций**.

Возможности репликации в различных выпусках SQL Server 2005

Все редакции SQL Server 2000 (за исключением SQL Server CE) поддерживали все типы репликации. В SQL Server 2005 ситуация несколько изменилась. Возможности репликации в различных редакциях SQL Server 2005 отражены в табл. 1.

Таблица 1. Типы репликации, поддерживаемые различными редакциями SQL Server (источник: BOL)

Возможности репликации	Выпуски Enterprise/ Developer Editions (32- и 64-разрядные)	Выпуск Standard Edition (32-разрядный)	Выпуск Workgroup Edition (32-разрядный)
Репликация слиянием	Да	Да	Да
Репликация транзакций	Да	Да	Да
Репликация моментальных снимков	Да	Да	Да
Подписчики, отличные от SQL Server	Да	Да	Нет
Публикация данных из Oracle	Да	Нет	Нет
Одноранговая репликация транзакций	Да	Нет	Нет

Подписчики, отличные от SQL Server, публикация данных из Oracle и одноранговая репликация транзакций – новые возможности репликации в SQL Server 2005, но они доступны не во всех выпусках. Как видно из таблицы, одноранговая репликация транзакций доступна только в Enterprise и Developer редакциях SQL Server 2005.

Выпуск SQL Server 2005 Developer Edition включает в себя всю функциональность выпуска Enterprise. Однако этот выпуск предназначен только для разработки и тестирования, и не может использоваться в эксплуатационной среде. Следовательно, для того, чтобы воспользоваться преимуществами одноранговой репликации транзакций, необходимо иметь выпуск Enterprise. Розничная цена процессорной лицензии для редакции Enterprise примерно в 4 раза выше, чем для редакции Standard, так что вопрос стоимости может быть одним из важных факторов в принятии решения.

Использование одноранговой репликации транзакций

Предположим, что у нас есть обычное приложение электронной коммерции. Для того чтобы избежать простоев и снизить нагрузку на отдельные серверы, база данных этого приложения располагается более чем в одном офисе. Поскольку это online-приложение и данные меняются (добавляются, изменяются, удаляются) в каждом офисе, то все модификации данных в одном офисе должны реплицироваться на серверы во всех остальных офисах. Предположим, что у нас есть базы данных в офисах A, B и C (рис. 1).

Чтобы реализовать подобную схему синхронизации данных в SQL Server 2000 мы должны использовать репликацию слиянием.

Офис A – издатель, офисы B и C – подписчики.

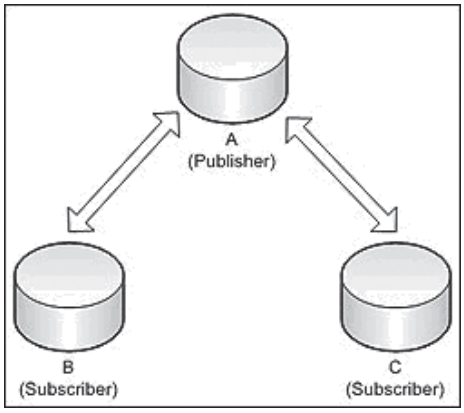


Рис. 1. Репликация слиянием

Очевидным недостатком такого метода является то, что в нем присутствует «единая точка отказа» в конфигурации репликации. Если B откажет, то репликация между A и C будет продолжать работать и пользователи, подключенные к серверам A и C, не заметят никаких осложнений в работе. Однако, если откажет узел A, то B и C будут «изолированы» от системы и изменения на них не будут видны.

Репликация SQL Server 2000 использует иерархию «издатель-подписчик». Успешное функционирование данной конфигурации требует, чтобы издатель был постоянно доступен.

В одноранговой (peer-to-peer) топологии репликации SQL Server 2005 каждый узел действует и как издатель, и как подписчик (рис. 2). Изменения (вставки, обновления и удаления) могут выполняться на всех узлах. Репликация распознает, что на заданном узле произошли изменения, но позволяет этим изменениям пройти по всем узлам только один раз.

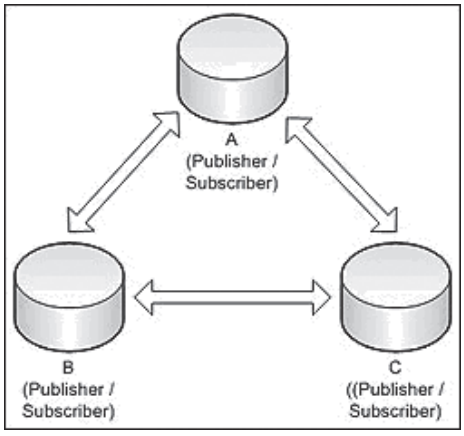


Рис. 2. Одноранговая репликация

Если один из узлов недоступен (например, A), остальные узлы (B и C) продолжают участвовать в репликации. Когда узел A снова становится доступен, он может синхронизироваться с другими узлами (B и C) и получить те изменения, которые произошли после его отказа. Это возможно благодаря тому, что все узлы (A, B и C) функционируют и как издатели, и как подписчики.

Как настроить одноранговую репликацию

Теперь, когда мы знаем как работает одноранговая репликация, давайте рассмотрим варианты ее настройки.

Для начала нам необходимо создать публикацию. Для этого используется мастер создания публикаций (New Publication Wizard).

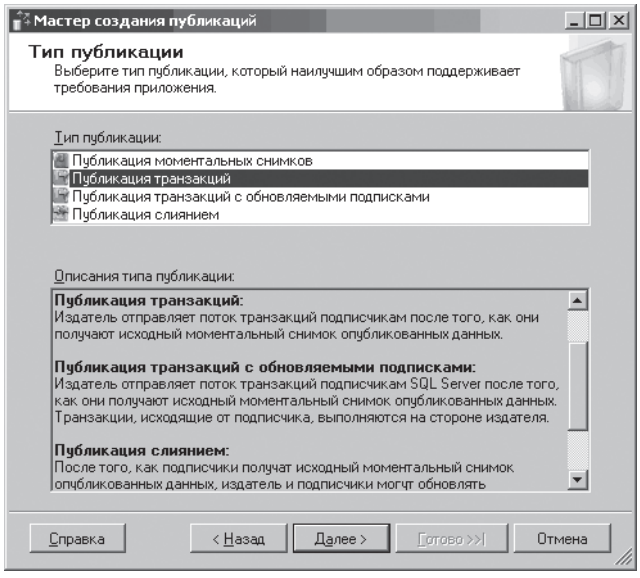


Рис. 3. Мастер создания публикаций

Как показано на рис. 3, существует 4 типа публикаций. Чтобы настроить одноранговую репликацию, необходимо выбрать «Публикацию транзакций» (Transactional publication).

После создания публикации необходимо изменить ее свойства. В окне «Свойства публикации» (рис. 4) установите значение параметра «Разрешить одноранговые подписки» в «True». Обратите внимание, после того как вы установили значение этого свойства в «True» вы не сможете вернуть его обратно в «False» до тех пор, пока репликация для этой публикации не будет удалена.

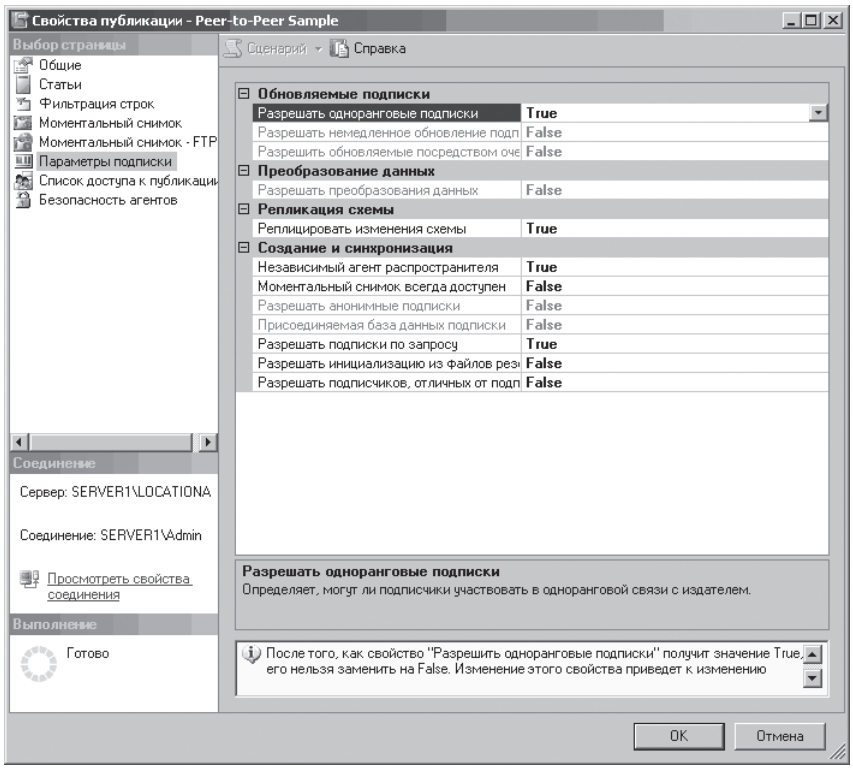


Рис. 4. Свойства публикации

Теперь нажмите правую кнопку мыши на публикации и запустите мастер транзакционной репликации (рис. 5).

При помощи этого мастера можно добавить дополнительные серверы или экземпляры SQL Server в сеть одноранговой репликации. Вы не сможете повторно добавить главные базы публикации или ранее выбранную базу данных, для которой мы запустили мастер.

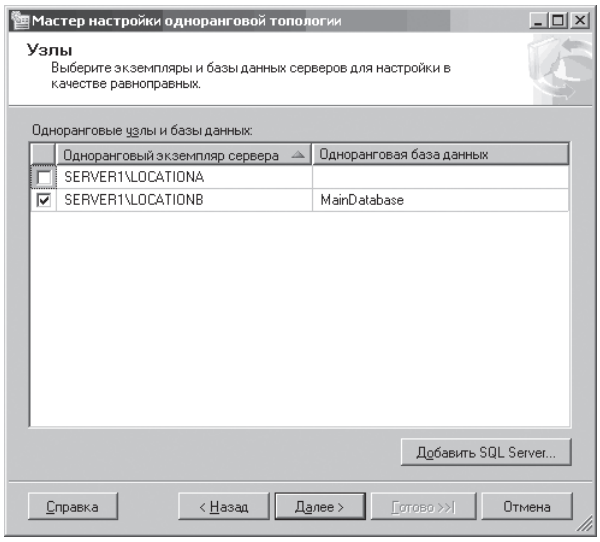


Рис. 5. Мастер настройки топологии: выбор узлов

После настройки сети репликации необходимо указать, каким образом будут инициализированы базы на других серверах в сети репликации.

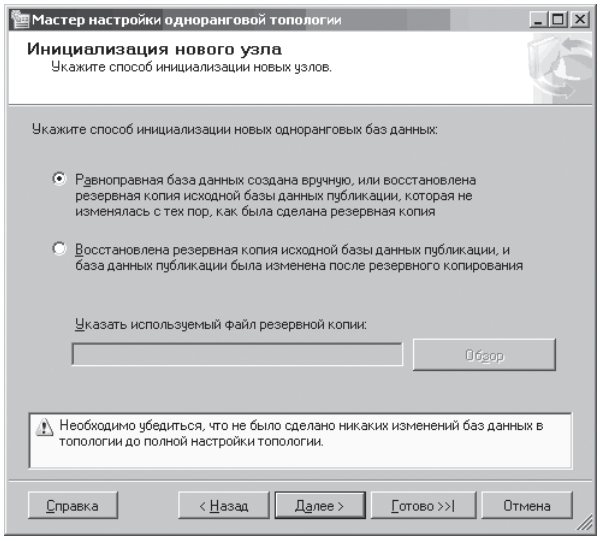


Рис. 6. Мастер настройки одноранговой топологии: инициализация нового узла

Наиболее очевидным путем инициализации новой базы является восстановление ее из резервной копии (необходимо удостовериться, что публикуемая база данных не менялась с тех пор, когда была сделана резервная копия). Если же со времени создания резервной копии в публикуемую базу были внесены изменения, мы можем выбрать второй пункт. В этом случае репликация доставит из первой базы данных публикации те изменения, которые не были включены в резервную копию.

После указания учетных данных для агента чтения журнала (Log Reader) и SQL Agent, мастер начнет построение одноранговой топологии. В нашем случае результат будет заключаться в создании трех публикаций и шести подписок (рис. 7). Для каждой публикации мастер также создаст базу данных распространителя.

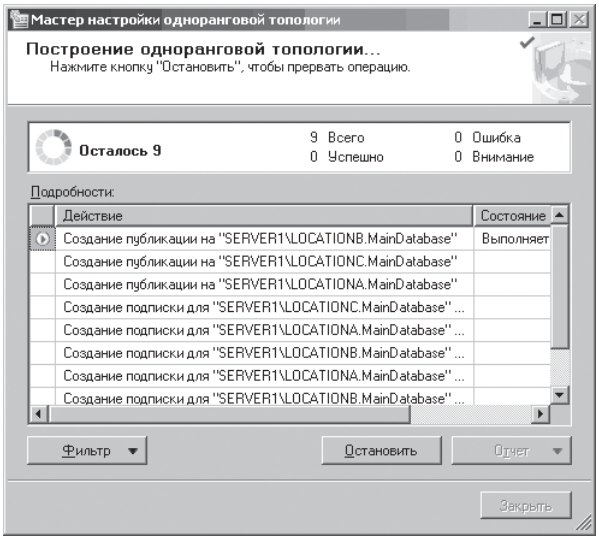


Рис. 7. Мастер настройки одноранговой топологии: создание топологии

По окончании работы мастера, на первом сервере мы получим следующий результат:

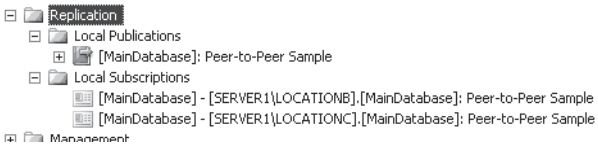


Рис. 8. Результат работы мастера

Добавление нового узла

А что если нам понадобится добавить еще один узел? Все, что нам нужно будет сделать в SQL Server 2005, используя одноранговую репликацию –

настроить новый узел как издателя и подписчика и присоединить его к одноранговой сети.

Заключение

Одноранговая репликация транзакций добавила новое измерение к репликации в SQL Server, она дала администраторам баз данных еще больше гибкости в синхронизации данных между несколькими серверами. Так как эта возможность новая, она не получила пока широкого распространения, но, несомненно, получит. Так что обратите на нее внимание, и посмотрите, какие преимущества она может предоставить вашей организации.

Совершенствование репликации в SQL Server 2008 (кодовое название Katmai)

В новой версии MS SQL Server 2008 (Katmai) произведены некоторые изменения, облегчающие настройку и управление одноранговой репликацией транзакций.

В частности, добавлена возможность подключения нового узла репликации в топологию одноранговой сети репликации без необходимости прекращения изменений на всех остальных узлах. Ранее для добавления нового узла необходимо было останавливать активность на всех узлах, после чего требовалось убедиться, что все незаконченные изменения успешно применены на всех узлах.

Кроме того, добавлена возможность визуальной конфигурации сети репликации. С помощью нового средства просмотра вы можете увидеть, как именно сконфигурирована топология сети, добавить или удалить узлы репликации, добавить или удалить связи между ними. Также в новом инструменте есть возможность создания связей репликации только между некоторыми узлами, в то время как старая форма представления топологии в виде таблицы требовала связи каждого узла с каждым.

Управление одноранговой репликацией в SQL Server 2005

По материалам статьи Дэвида Линдквиста (David Lindquist) «Managing SQL Server 2005 Peer-to-Peer Replication»

Перевод Владислава Александровича Щербинина

Корпоративные веб-сайты Microsoft очень существенно зависят от баз данных, которые обслуживают такие сайты с высоким трафиком, как Microsoft® Update, Download Center, Communities, TechNet и MSDN®. В отделе эксплуатации Microsoft.com есть команда из 17 инженеров — команда обслуживания SQL Server™, которая управляет этими системами баз данных. В общей сложности эта команда отвечает за работоспособность более чем 2250 активных баз данных, которые занимают около 55.15TB на 291 сервере SQL Server.

Одна из систем, обслуживаемая этой командой, содержит базы данных для внутренних контент-провайдеров. Эти контент-провайдеры представляют собой группы в Microsoft, публикующие информацию в Веб. Они расположены по всему миру и всем им требуются функции баз данных, которые обслуживают их веб-приложения.

Данная среда имела различную архитектуру по мере изменения баз данных, функциональных требований и стратегий хостинга. В настоящее время эта среда объединяет более чем 100 баз данных, работающих под управлением Windows Server™ 2003 SP1 Enterprise Edition и SQL Server 2000 SP4 на шести серверах HP Proliant. Транзакционная репликация используется для перемещения данных с и на консолидирующий сервер, который предоставляет данные множеству серверов, к которым обращаются веб-приложения. Также для обеспечения возможности восстановления после отказов и создания резервных серверов используется технология доставки журналов транзакций. На рис. 1 показана эта архитектура.

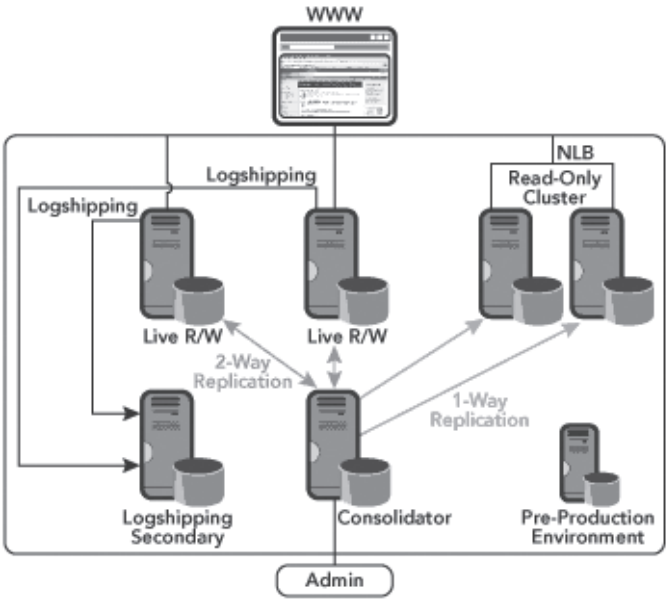


Рис. 1. Текущая архитектура

С приближением выхода SQL Server 2005, мы заинтересовались несколькими новыми возможностями этой системы, и нам требовалось разработать план обновления текущей среды. Одна из этих возможностей, которая нас особенно интересовала, это новый тип репликации транзакций под названием одноранговая репликация (peer-to-peer replication). В такой конфигурации все узлы в топологии равноправны. Каждый узел публикует и подписывается на одну и ту же схему и данные. Изменения (такие как вставки, изменения и удаления) могут выполняться на любом узле, и эта технология репликации распознает, что изменения были применены к заданному узлу, предотвращая прохождение по узлам более одного раза.

В статье SQL Server Books Online «Replication Availability Enhancements» (Улучшенная доступность репликации) говорится, что «в SQL Server 2000 репликация транзакций поддерживала иерархические топологии, в которых издатель владеет данными, реплицируемыми подписчикам. Репликация транзакций с обновляемыми подписчиками поддерживала выполнение изменений на стороне подписчика, но подписчики и издатели считались разными типами участников репликации».

В этой статье будет рассказано о новой одноранговой модели репликации в SQL Server 2005, которая позволяет выполнять репликацию между равноправными участниками топологии. Эта модель хорошо подходит для межсерверных конфигураций в которых также требуется динамически изменять роли узлов для выполнения обслуживания или для восстановления после сбоев.

Мысль о том, чтобы иметь одну и ту же базу данных на более чем одном сервере, и при этом все базы синхронизированы друг с другом, и каждая из них доступна и на запись, и на чтение, была замечательной. Мы могли бы балансировать нагрузку на базы данных, когда не выполняется обслуживание, а когда требуется выполнить некоторые работы на сервере, мы могли бы исключить его из работы, и другие серверы взяли бы на себя нагрузку. Время доступности, производительность и возможности восстановления после сбоев были довольно заманчивыми.

Сравнение стратегий балансировки нагрузки и восстановления после отказов

Стратегия I: Характеристики балансировки нагрузки

Каждый сервер обслуживает копию базы данных, доступную для чтения и записи. Базы данных синхронизируются с помощью одноранговой репликации SQL Server 2005. Приложения соединяются с кластером через специальный интерфейс, используя имя хоста, который будет распределять и балансировать трафик между четырьмя узлами.

Избыточность (резервирование)

Копия базы данных существует на четырех разных серверах. Если один из узлов кластера становится недоступен, остальные узлы автоматически принимают трафик.

Доступность

Каждый из серверов может быть изъят из кластера, таким образом можно выполнять обслуживание, и база данных не станет недоступной.

Производительность

Запросы приложения к базе данных распределяются между четырьмя узлами в кластере. Балансировка нагрузки должна улучшить производительность во время повышенной активности.

Стратегия II: Характеристики восстановления после отказов

Каждый сервер обслуживает копию базы данных, доступную для чтения и записи. Базы данных синхронизируются с помощью одноранговой репликации SQL Server 2005. Приложения подключаются к одному узлу кластера, используя один из виртуальных IP-адресов этого кластера.

Избыточность (резервирование)

В случае если узел, принимающий трафик становится недоступным, или если требуется выполнить обслуживание сервера, трафик перенаправляется на другой узел кластера. Для обеспечения избыточности текущая копия базы данных существует на всех четырех узлах.

Доступность

Если узел кластера, к которому выполняется подключение, становится недоступным, или его требуется изъять для обслуживания, трафик перенаправляется на другой узел кластера с минимальным временем простоя (несколько секунд).

Производительность

Приложение всегда подключается к одной базе данных в топологии.

Наша стратегия обновления

Проект обновления мы начали с установки в нашей лаборатории двух тестовых серверов. Каждый из них работал под управлением Windows Server 2003 SP1 и имел два экземпляра SQL Server 2005. В результате мы получили четыре узла для тестирования. Затем на каждом экземпляре мы установили тестовую базу данных и настроили одноранговую репликацию. Мы обнаружили, что можем изменять данные и схему в любой из четырех баз данных, и эти изменения без проблем реплицируются на другие узлы. Мы создали начальный прототип. Распространение изменений схемы было замечательной возможностью. Теперь мы можем изменять таблицы, представления, процедуры, функции и триггеры (только DML-триггеры), и изменения будут реплицированы на остальные узлы.

Так как в центре обработки данных Microsoft используются 64-разрядные технологии (и учитывая возраст внутренних серверов на тот момент), мы решили приобрести новые серверы в конфигурации, показанной в табл. 1. (За более подробной информацией о нашем переходе на 64-разрядные технологии, обратитесь к статье в January/February 2006 Inside Microsoft.com column.

Таблица 1. Новые серверы

	Модель	ОС	Процессоры	Память	Приложение
Эксплуатационная среда	64-bit HP Proliant DL585 G1	Windows Server 2003 Enterprise x64 Edition SP1	Четыре одноядерных процессора AMD Opteron 2.20GHz	16GB	SQL Server 2005 SP1
Предэксплуатационная среда	64-bit HP Proliant DL385 G1	Windows Server 2003 Enterprise x64 Edition SP1	Два двуядерных процессора AMD Opteron 2.21Ghz	8GB	SQL Server 2005 SP1

Балансировка нагрузки

Четыре сервера включены в два кластера для веб-серверов Microsoft.com с помощью балансировки сетевой нагрузки (Network Load Balancing,

NLB). Два других сервера установлены в центре обработки данных для предэксплуатационной среды. Мы используем предэксплуатационную среду (pre-production environment, PPE), чтобы наши пользователи, провайдеры контента, могли установить свои приложения на серверах, имитирующих эксплуатационные серверы. Таким образом они могут выполнять разработку и тестирование в среде, которая обеспечивает такие же версии операционных систем и приложений, на которых приложения будут работать в эксплуатационной среде. На рис. 2 показана эта конфигурация.

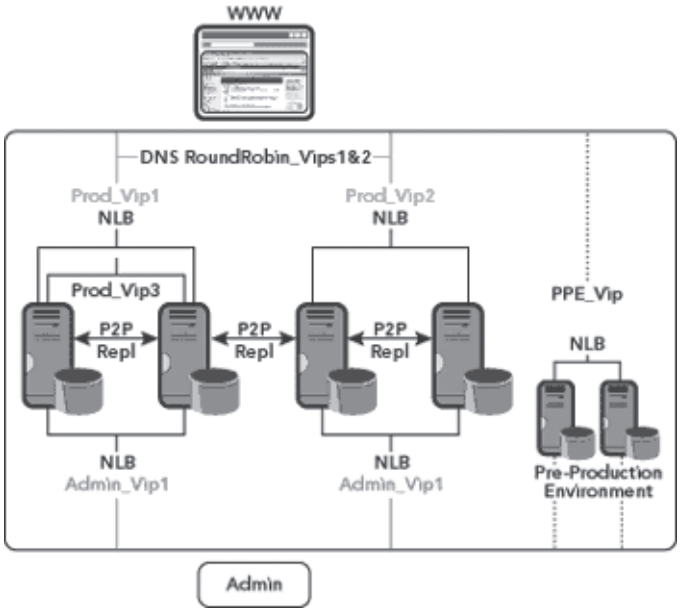


Рис. 2. Среда с балансировкой нагрузки

Каждый сервер имеет две сетевые карты. Административная сетевая карта используется владельцами базы данных для манипуляции данными. Мы предоставили им разрешение db_datareader в базе данных и разрешения для вставки, модификации и удаления на отдельные таблицы, чтобы они могли загружать и выгружать данные из эксплуатационной базы данных не обращаясь для этого к нам. Эксплуатационная сетевая карта обрабатывает входящий и исходящий трафик от веб-серверов, а также трафик репликации между узлами.

Для эксплуатационных серверов мы настроили пару двухузловых кластеров. Мы добавили виртуальный IP-адрес для каждой пары административных сетевых карт и еще один виртуальный IP-адрес для каждой пары эксплуатационных сетевых карт. Это позволяет нам управлять трафиком с обеих сторон кластеров — с эксплуатационной стороны и с административной стороны. Мы создали DNS-имя хоста, которое рас-

пределяется между двумя эксплуатационными виртуальными IP-адресами, так что мы можем использовать все четыре узла для балансировки нагрузки.

Мы настроили NLB в режиме unicast (адресация конкретному устройству), и воспользовались преимуществом новой возможности Windows Server 2003 SP1, которая позволяет двум серверам, настроенным для балансировки нагрузки, взаимодействовать друг с другом, используя интерфейс NLB. За более подробной информацией об этой возможности обратитесь к статье Базы Знаний «Unicast NLB nodes cannot communicate over an NLB-enabled network adaptor in Windows Server 2003».

При тестировании балансировки нагрузки стали очевидны внутренние задержки в системе (то есть время между фиксацией транзакции на издателе и фиксацией соответствующей транзакции на подписчике). У нас не было уверенности в том, что репликация с одного узла на три остальные будет занимать фиксированное время. Это означало, что требуется уточнить некоторые детали решения, связанные с проектированием базы данных.

Замечания о решении

При использовании балансировки нагрузки не гарантируется, что при обращении приложения к базе данных оно каждый раз будет обращаться к той же самой базе данных. Мы обнаружили, что балансировка нагрузки хорошо работает для баз данных, используемых только для чтения, и для некоторых баз данных, используемых только для записи. Однако могут возникнуть проблемы, если приложению требуется поддерживать любой тип состояния сеанса. При использовании балансировки нагрузки мы не могли гарантировать, что каждый раз, когда приложение обращается к кластеру, оно обратится к той же самой базе данных. И что еще хуже, так как веб-серверы также объединены в кластер, мы не могли гарантировать, что последующие обращения приложения будут исходить от одного и того же веб-сервера.

Это может привести к проблемам, например, если приложению требуется читать данные сразу же после того, как они были введены. Скажем, приложение сохраняет изменение, выполненное пользователем, и сразу же читает эти данные, чтобы отобразить их пользователю. Если приложение подключается к одной базе данных для сохранения данных, но подключается к другой для их чтения, вторая база данных может еще не иметь измененных данных (потому что было недостаточно времени для репликации), и пользователю будут возвращены старые данные.

Мы должны были придумать некоторый способ, чтобы приложение гарантированно обращалось к одной и той же базе данных, но при этом удовлетворить требование доступности. После некоторых исследований мы обнаружили, что можно создать множество виртуальных IP-адресов

для пары интерфейсов с NLB и управлять этими IP-адресами независимо от других IP-адресов, настроенных для тех же самых интерфейсов. Это означало, что мы можем создать второй эксплуатационный виртуальный IP-адрес для одного из кластеров. Перенаправив 100 процентов трафика с этого IP-адреса на один из узлов кластера, мы могли бы гарантировать, что приложение будет обращаться к одной и той же базе данных при каждом обращении к этому IP-адресу. Это дало нам гибкость при управлении трафиком, используя этот виртуальный IP-адрес между двумя узлами кластера, и позволило нам переключаться с одного узла на другой в случае отказа.

Задержка репликации также может повлиять на ограничения внешних ключей. Например, представьте, что у нас есть внешний ключ, которому требуется значение в столбце первичного ключа в некоторой записи таблицы Table 1, прежде чем значение внешнего ключа может быть вставлено в таблицу Table 2. Если значение вставлено в столбец первичного ключа таблицы Table 1 в базе данных DB1, но это значение еще не было реплицировано в таблицу Table 1 базы данных DB2, и приложение попытается вставить значение в столбец внешнего ключа таблицы Table 2 базы данных DB2, ему это не удастся, так как необходимое значение первичного ключа еще не существует. Конечно, если все это происходит в одном подключении к базе данных, не никаких проблем. Но если это происходит в отдельных вызовах с веб-сервера через несколько секунд, существует потенциальная возможность ошибки.

Если таблица содержит идентифицирующие столбцы (identity), определенные как первичные ключи, необходимо принимать во внимание некоторые детали, как для балансировки нагрузки, так и для восстановлений после сбоев. Например, когда запись из таблицы издателя со значением identity реплицируется в таблицу подписчика, и свойство Not For Replication установлено для этого столбца на подписчике, значения записи вставляются в таблицу на подписчике, но вставка не увеличивает значение identity. В результате, когда приложение попытается добавить запись непосредственно в таблицу на подписчике (который теперь может являться издателем), оно потерпит неудачу, так как уже существует запись в столбце identity со следующим значением identity, так как оно было реплицировано с первоначального издателя.

Проблем со столбцами identity можно решить двумя способами. Во-первых, в таблице, содержащей столбец identity, можно создать новый столбец. Этот столбец будет иметь целочисленный тип данных (что добавит 4 байта к записи) и по умолчанию будет содержать идентификатор сервера, на котором расположена база данных. Например, если идентификатор сервера для первого узла равен 1, для второго — 2, и так далее, новый столбец на первом узле будет по умолчанию иметь значение 1, новый столбец на втором узле будет по умолчанию иметь значение 2, и так далее. Этот новый столбец, или столбец SRVID, и столбец identity будут настроены как составной первичный ключ. Это в конечном итоге

гарантирует уникальность значений ключа на всех узлах кластера, и решает проблему с нарушением уникальности ключей.

Второй вариант состоит в том, чтобы задать значениям identity отдельные диапазоны. Это гарантирует, что значения identity будут уникальны во всех четырех экземплярах базы данных. Для снижения усилий по поддержке такого варианта, столбцы identity создаются с использованием типа данных bigint. Мы используем bigint, чтобы можно было создавать большие диапазоны (миллиард), которые реже требуется изменять. Например, диапазон для первого узла начинается с 1, для второго узла — с 1,000,000,001, диапазон третьего узла начинается с 2,000,000,001, и диапазон четвертого узла начинается с 3,000,000,001. В таком варианте также добавляется 4 байта к записи, так как тип bigint занимает 8 байт вместо четырех для типа int.

Эксплуатация

Прелесть этой системы состоит в гибкости, которая позволяет нам обслуживать множество баз данных в различных конфигурациях. Мы можем использовать преимущество того факта, что у нас есть четыре сервера, и мы можем распределить нагрузку по всем четырем серверам, или по двум или по трем, или даже перенаправить трафик только на один сервер.

Подключение к большинству баз данных в этой среде выполняется посредством имени источника данных (data source name, DSN), которое настроено в реестре на веб-серверах. С помощью DSN мы управляет трафиком. У нас есть специальный сценарий, который изменяет DSN на веб-серверах. Если DSN указывает на имя хоста, который выполняет балансировку нагрузки между всеми четырьмя серверами, мы можем запустить сценарий для изменения DSN, последовательно на каждом сервере, чтобы перенаправить его на один узел кластера. Затем мы можем выполнить любые изменения в базе данных и проверить, что изменения успешно реплицированы на остальные три узла, прежде чем изменять DSN на имя хоста, выполняющего балансировку нагрузки между узлами кластера. К тому же, если по какой-то причине возникла проблема с репликацией в тот момент, когда DSN указывает на один узел, мы можем удалить репликацию и восстановить ее, прежде чем настроить DSN на остальные три узла. Мы можем делать все это в то время как сайты продолжать успешно обращаться к базе данных.

В стратегии восстановления после отказов, приложение и так работает с одной базой данных. Так что мы можем выполнять изменения на этом одном узле и не беспокоиться о доступности других узлов, так как приложение не обращается к ним непосредственно.

Следует помнить о том, что эта среда может очень быстро стать очень сложной. Для каждой базы данных существует публикация и три подписки, по одной для каждого из трех остальных узлов. Это означает, что

существует четыре копии базы данных (по одной для каждого узла), четыре публикации (по одной для каждого узла), и двенадцать подписок для каждой базы данных (три на каждом узле). Таким образом, количество заданий и процессов, работающих на серверах, может существенно увеличиться. Например, если у нас есть 100 различных баз данных, добавится 400 баз данных, 400 публикаций и 1200 подписок.

Множество виртуальных IP-адресов для сетевых интерфейсов также может создавать сложности. Если подключения интерфейсов не очень хорошо задокументированы и не очень ясны, виртуальный IP-адрес для восстановления после сбоев может управляться неправильно, что приведет к тому, что группа баз данных станет недоступной.

Значения identity это одна из самых больших проблем, которую надо решить. Проблемы возникают из-за того, что многие разработчики используют значение identity для идентификации логической сущности, например бизнес-имени, имени контакта или названия товара. Это может привести к незапланированному времени разработки, которое потребуется для адаптации существующего приложения для использования составного первичного ключа, который будет включать новый столбец-идентификатор сервера и столбец identity. Мы считаем, что это стоит сделать, но если столбец-идентификатор сервера нельзя добавить, потребуется определить и управлять диапазонами значений identity. Один из способов управления диапазонами состоит в создании задания SQL Server Agent, которое будет отслеживать значения столбцов identity в таблицах баз данных. Если это значение достигнет некоторого порога, будет запущена хранимая процедура, которая изменит начальное значение identity.

Нам была очень важна возможность восстановления репликации после сбоя. Если данные в базах данных станут несогласованными, мы должны перенаправить трафик на один узел и затем определить и устранить различия между таблицами на остальных узлах. С SQL Server 2005 поставляется новая утилита под названием tablediff.exe. Она очень полезна для определения различий между таблицами и создания сценариев для корректировки этих различий. После того как все данные будут восстановлены в одной базе данных, можно перенастроить репликацию и перенаправить DSN назад на кластер.

Теперь мы успешно используем одноранговую репликацию для различных источников данных в Microsoft. В дальнейшем мы планируем внедрить решение с использованием географического кластера, чтобы еще больше увеличить гибкость и надежность этой инфраструктуры.

Изменения в репликации транзакций: трассировочные маркеры

По материалам статьи Пола Ибизона (Paul Ibison)
«Transactional Changes: 1 - Tracer Tokens»
Перевод Яна Дмитриевича Либермана

Введение

В репликацию транзакций в SQL Server 2005 была добавлена поддержка трассировочных маркеров (tracer token). Благодаря этой новой функциональности у нас появился простой способ измерения задержек в различных схемах репликации транзакций. Первоначально трассировочные маркеры записываются в журнал транзакций базы данных публикации. Далее они обрабатываются как обычные транзакции, то есть проходят полный путь от издателя к подписчику. В действительности на подписчике ничего не применяется, но, тем не менее, этот метод позволяет точно определить задержки в цепи «издатель > распространитель > подписчик». Чтобы воспользоваться этой полезной функциональностью, издатель и распространитель должны быть SQL Server 2005 (поддерживается также издатель Oracle). Для принудительных подписок подписчиком может быть SQL Server 2000. А для подписок по запросу полная статистика собирается только, если подписчик SQL Server 2005, в противном случае измеряются только задержки в цепи «издатель > распространитель»

Реализация

(a) Используя графический интерфейс

Запустите монитор репликации, и в списке слева выберите нужную публикацию (для примера, приведенного на рис. 1, это PubDocs). В правой части окна доступны три закладки. Первая закладка относится к агентам распространителя, а последняя – к агентам моментального снимка и агенту чтения журнала. Средняя закладка открывает доступ к функциональности трассировочных маркеров. Чтобы добавить новый трассировочный маркер, нажмите кнопку «Insert New Tracer...». После этого трассировочный маркер появится в таблице (под кнопкой). Так как

трассировочный маркер был только что добавлен в журнал транзакций издателя и еще не начал свое движение в сторону подписчика, во всех столбцах кроме «Subscription» (подписка) будет отображено значение «Pending...». Эта таблица обновляется автоматически каждые 10 секунд. Но по моему опыту, трассировочные маркеры могут ошибочно оставаться в состоянии «Pending...» даже после обновления. Выполнение обновления по требованию в левой панели, всегда отображает правильное состояние.

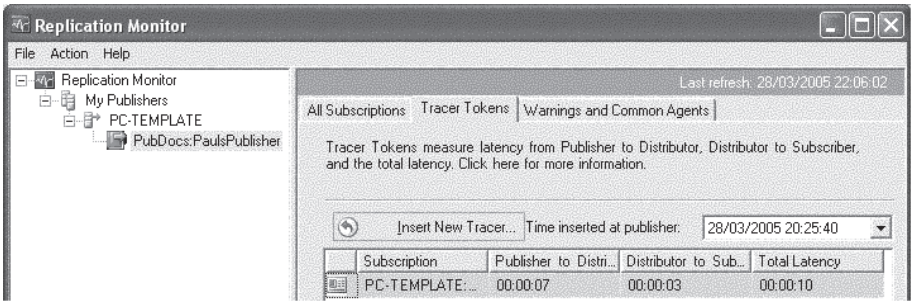


Рис. 1.

(b) Используя TSQL

Чтобы добавить трассировочный маркер «вручную», нужно воспользоваться процедурой sp_posttracertoken:

```
DECLARE @myTokenID AS int
EXEC sp_posttracertoken
  @publication = 'pubdocs',
  @tracer_token_id = @myTokenID OUTPUT;
select @myTokenID
```

Совет. Параметр @tracer_token_id процедуры sp_posttracertoken не является обязательным, но если вы хотите получить статистику по конкретно этому трассировочному маркеру, то значение, которое возвращается с помощью этого параметра, позволяет его идентифицировать.

Процедура sp_helptracertokenhistory возвращает по одной строке для каждого подписчика с информацией о задержках для указанного трассировочного маркера:

```
EXEC sp_helptracertokenhistory
  @publication = 'pubdocs',
  @tracer_id = @myTokenID
```

Примечание. Эта процедура возвращает задержку на этапе «распространитель > подписчик» (столбец subscriber_latency) и полную задержку (столбец overall_latency). Хотя задержка на этапе «издатель > распространитель» не возвращается, ее можно легко получить как overall_latency - subscriber_latency.

Для получения списка всех трассировочных маркеров, можно выполнить запрос к таблице MStracer_tokens базы данных распространителя:

```
select * from distribution..MStracer_tokens
```

Ту же информацию можно получить и при помощи процедуры sp_browsereplcmds. Строки с типом, равным 1073741871, и есть трассировочные маркеры.

Инициализация подписки без первоначальной синхронизации данных в SQL Server 2005

По материалам статьи Пола Ибизона (Paul Ibison) «NoSync Initializations (and variations) on SQL Server 2005»

Перевод Яна Дмитриевича Либермана

Я хочу представить вам небольшую статью про инициализацию подписок без первоначальной синхронизации данных в SQL Server 2005[1]. Я уже писал на эту тему в контексте SQL Server 2000, однако в новой версии предлагаются новые подходы, о которых, безусловно, стоит рассказать.

Добавление подписки с помощью мастера

Раздел «Initialize Subscriptions» мастера добавления подписки изменился в SQL Server 2005. Для того чтобы отказаться от автоматической инициализации, вместо выбора опции «No, the Subscriber already has the schema and data» теперь нужно сбросить соответствующий флажок в столбце «Initialize» (рисунок 1). Подход к инициализации подписки на публикацию слиянием без моментального снимка практически не изменился. В процессе начальной синхронизации агент слияния создает системные таблицы, триггеры и другие необходимые объекты. Однако инициализация подписки на публикацию транзакций без моментального снимка, осуществляется не так, как это было в SQL Server 2000. Для этого используется новая опция «replication support only», которая рассматривается далее.

[1] Инициализацию подписки без первоначальной синхронизации данных часто называют «nosync» инициализацией. Название пошло от значения параметра @sync_type = 'none' процедуры sp_addsubscription

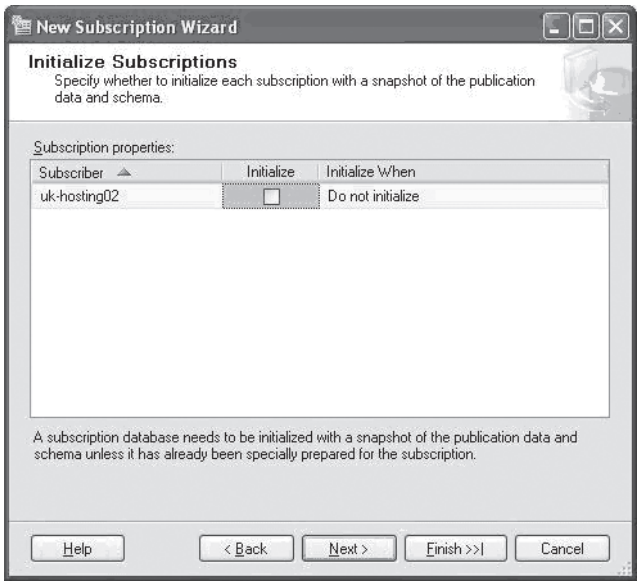


Рис. 1.

Добавление подписки с помощью TSQL

К сожалению, возможности мастера добавления подписки сильно ограничены. Только при настройки репликации «вручную» непосредственно с помощью TSQL нам доступны все возможности репликации без ограничений, которые накладывает графический интерфейс. Чтобы добавить подписку в репликации транзакций, используются следующие процедуры:

- sp_addsubscription
- sp_addpushsubscription_agent \ sp_addpullsubscription_agent

В SQL Server 2000 для параметра @sync_type процедуры sp_addsubscription было предусмотрено только два значения: «automatic» (значение по умолчанию) и «none» (значение, которое использовалось для инициализации подписок без первоначальной синхронизации данных). Надо сказать, что это по-прежнему справедливо для репликации слиянием. Поэтому далее речь пойдет исключительно о репликации транзакций. В SQL Server 2000, если использовался тип инициализации «none», то нужно было вручную создать на подписчике пользовательские хранимые процедуры (используя sp_scriptpublicationcustomprocs). В SQL Server 2005 у нас есть альтернатива. Так, к двум уже существующим типам добавлено еще два новых типа синхронизации: «replication support only» и «initialize with backup». Итак, список всех поддерживаемых в SQL Server 2005 типов инициализации:

1. **«none»** – предполагается, что подписчик уже имеет схему и начальные данные для опубликованных таблиц. Этот аргумент является устаревшим. Вместо него следует использовать «replication support only» (интересно, что процедура sp_scriptpublicationcustomprocs не помечена в электронной документации по SQL Server 2005 (BOL) как устаревшая для ручного использования).
2. **«automatic»** (значение по умолчанию) – схема и начальные данные для опубликованных таблиц передаются подписчику автоматически.
3. **«replication support only»** – что означает термин «replication support» (поддержка репликации) в данном контексте? Это хранимые процедуры, триггеры, которые поддерживают обновляемые подписки, а также необходимые системные таблицы. То есть, все аналогично SQL Server 2000 (тип «none»), за исключением необходимости вручную создавать на подписчиках все эти вспомогательные объекты. Этот вариант, как и «none», предполагает, что подписчик уже имеет схему и начальные данные для опубликованных таблиц. Мы должны быть очень внимательны - данные на издателе и подписчике должны быть абсолютно идентичны. В процессе настройки выполнение этого условия не проверяется и, следовательно, ответственность за его выполнение полностью лежит на нас – администраторах репликации. Другими словами, нужно зафиксировать систему (quiescing the system). Очевидно, что если ваша система работает в режиме круглосуточного доступа, то сделать это невозможно. В этом случае следует использовать варианты (2) и (4).
4. **«initialize with backup»** – схема и начальные данные для опубликованных таблиц извлекаются из резервной копии базы данных публикации. Соответственно, мы должны обеспечить подписчику доступ к последней резервной копии базы данных публикации – то есть либо переправить ее каким-то образом на подписчика, либо поместить ее на доступный подписчику общий ресурс. Для указания местоположения резервной копии и типа носителя для резервной копии предусмотрены два новых параметра процедуры sp_addsubscription: @backupdevicename и @backupdevicetype.

После настройки подписок, инициализируемых с помощью резервной копии, нужно сделать три вещи:

1. Удалить лишние объекты и таблицы. Нет смысла засорять базу данных подписки, тем, что не было бы перенесено туда в случае автоматической инициализации. Это ускорит резервное копирование и сделает предназначение этого подписчика более прозрачным для других администраторов.
2. Заменить столбцы «timestamp» на «binary(8)». Для выполнения этой операции вам потребуется промежуточная таблица.

3. Помните, что резервная копия включает в себя все данные, даже если к реплицируемым таблицам применены фильтры строк или столбцов. Поэтому некоторые столбцы, возможно, также нужно будет удалить.

Как работает «initialize with backup»

При использовании типа инициализации «initialize with backup» не требуется фиксировать топологию репликации на время настройки. При использовании «initialize with backup», процедура sp_addsubscription в процессе работы выполняет команду RESTORE HEADERONLY для получения последнего регистрационного номера транзакции в журнале (LSN) в резервной копии для использования в качестве точки начала синхронизации. Далее это значение сравнивается со значениями столбца xact_seqno таблицы distribution..MSrepl_commands, после чего выбранные команды реплицируются на подписчика. Для того чтобы все необходимые команды присутствовали в таблице MSrepl_commands, резервная копия должна быть восстановлена до момента, когда будет достигнут максимальный срок хранения публикации. В противном случае нужные транзакции могут быть удалены задачей очистки на распространителе. Если это случится, то будет выдано следующее сообщение:

```
Msg 21397, Level 16, State 1, Procedure sp_MSsetupnosyncsubwithlsnatdist, Line 213
The transactions required for synchronizing the nosync subscription created from the
specified backup are unavailable at the Distributor. Retry the operation again with a more
up-to-date log, differential, or full database backup. The Subscriber was dropped.
```

Для того чтобы защититься от этой проблемы, электронная документация по SQL Server 2005 (BOL) рекомендует на этот период временно отключить задачу очистки на распространителе.

Заключение

Надеюсь, что эта статья поможет вам извлечь максимум пользы из новых возможностей при инициализации подписок без первоначальной синхронизации данных. Графический интерфейс не позволяет в полной мере управлять типами инициализации подписки. Поэтому вам, скорее всего, придется использовать TSQL. Но дополнительные возможности, которые при этом можно получить, представляют собой хорошую альтернативу автоматической инициализации.

Хочу поблагодарить Nigel Maneffa, за ряд ценных дополнений, которые позволили сделать эту статью лучше.

Репликация DML

По материалам статей Энди Уоррен (Andy Warren) «Replication Statement Delivery Options - Part 1» и «Replication Statement Delivery Options – Part 2»

Перевод Александра Сергеевича Волока

Одними из самых полезных свойств репликации транзакций является обилие конфигурационных настроек, в частности тех, что управляют передачей изменений подписчикам. Прежде чем начать изучение способов применения этих опций, мы должны понять, как работают значения по умолчанию этих опций, на чем, собственно и сфокусируется пока наше внимание.

В этой статье применяется очень простая база данных с именем «ReplTest», имеющая одну таблицу по имени «SomeNewTable», публикацию репликации транзакций с именем «Test» и одного подписчика. Нашей отправной точкой станет диалоговое окно свойств публикаций с выбранной вкладкой «Articles» и отмеченной таблицей «SomeNewTable». Затем из списка свойств публикаций следует выбрать пункт «Set Properties of Highlighted Table Article».

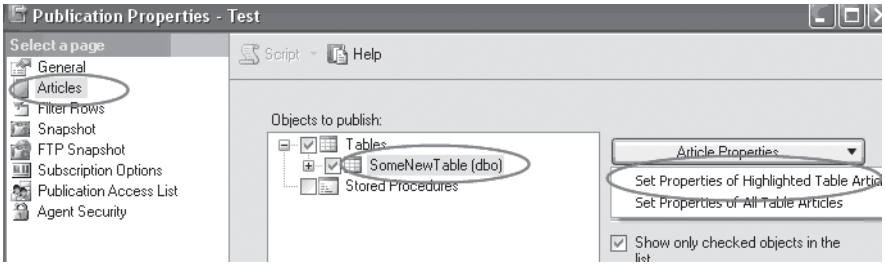


Рис. 1.

По умолчанию, результатом описанных выше действий будет появление диалогового окна с опциями доставки. Все эти опции указывают лишь на хранимые процедуры, уже созданные на подписчике в процессе применения снимка. В случае же наличия нескольких подписчиков, эти опции относятся к каждому из них. Нет официально-поддерживаемой возможности подстраивать доставку по отдельности для какждого из подписчиков, кроме как создавать отдельные публикации.

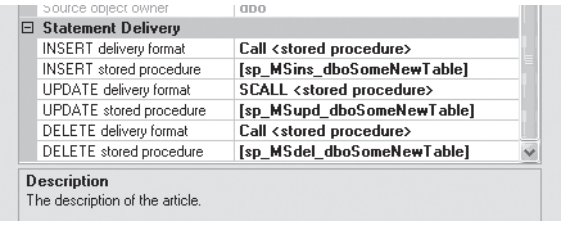


Рис. 2.

Прежде чем начать изменение этих опции, стоит взглянуть на уже созданные хранимые процедуры подписчика.

Процедура вставки

```
ALTER procedure [dbo].[sp_MSins_dboSomeNewTable]
  @c1 varchar(150),@c2 varchar(50),@c3 varchar(50)
as
begin
insert into [dbo].[SomeNewTable](
 [EmailAddress]
,[FirstName]
,[LastName]
 )
values (
 @c1
,@c2
,@c3
 )
end
```

Процедура обновления

```
ALTER procedure [dbo].[sp_MSupd_dboSomeNewTable]
 @c1 varchar(150) = null,@c2 varchar(50) = null,@c3 varchar(50) = null,@pkc1
varchar(150)
,@bitmap binary(1)
as
begin
if ( substring(@bitmap,1,1) & 1 = 1 )
begin
update [dbo].[SomeNewTable] set
 [EmailAddress] = case substring(@bitmap,1,1) & 1 when 1 then @c1 else
[EmailAddress] end
,[FirstName] = case substring(@bitmap,1,1) & 2 when 2 then @c2 else [FirstName] end
,[LastName] = case substring(@bitmap,1,1) & 4 when 4 then @c3 else [LastName] end
where [EmailAddress] = @pkc1
if @@rowcount = 0
    if @@microsoftversion>0x07320000
        exec sp_MSreplraiserror 20598
end
else
```

```
begin
update [dbo].[SomeNewTable] set
 [FirstName] = case substring(@bitmap,1,1) & 2 when 2 then @c2 else [FirstName] end
,[LastName] = case substring(@bitmap,1,1) & 4 when 4 then @c3 else [LastName] end
where [EmailAddress] = @pkc1
if @@rowcount = 0
    if @@microsoftversion>0x07320000
        exec sp_MSreplraiserror 20598
end
end
```

Процедура удаления:

```
ALTER procedure [dbo].[sp_MSdel_dboSomeNewTable]
  @pkc1 varchar(150)
as
begin
delete [dbo].[SomeNewTable]
where [EmailAddress] = @pkc1
if @@rowcount = 0
    if @@microsoftversion>0x07320000 exec sp_MSreplraiserror 20598
end
```

Самой простой из этих процедур является процедура вставки, которая содержит лишь выражение вставки. Можно убедиться, что в ней даже нет проверок на ошибки. И если при выполнении вставки, по какой-либо причине (дубликаты строк, несовпадение столбцов и т.п.) возникнет ошибка, выполнение агента распространителя будет остановлено с выдачей соответствующего предупреждения об ошибке. При взгляде на выражение удаления можно заметить, что выполняется простое удаление по первичному ключу с последующей проверкой количества затронутых строк (rowcount) – и если количество строк равно нулю или же при выполнении операции возникла любая другая ошибка, агент распространителя предупреждает об ошибке. Процедура обновления является самой сложной. Параметрами этой процедуры выступают новые значения полей таблицы и битовая карта, содержащая список полей, которые были изменены. Сперва выполняется ветвление по условию - был ли обновлен первичный ключ. Затем, исходя из полученной битовой карты, каждому полю таблицы устанавливается либо новое значение, либо текущее и выполняется обновление. И снова, если возникает какая-либо из ошибок, включая и отсутствие обновляемой строки на подписчике, агент распространителя завершит свою работу предупреждением об этой ошибке.

Ошибка, вызванная отсутствием искомой записи на подписчике:

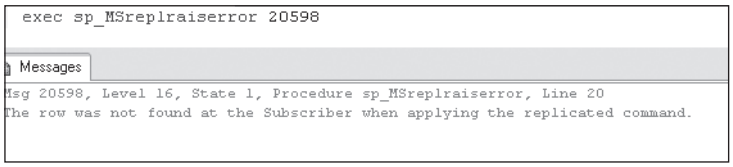


Рис.3.

Также стоит обратить внимание еще на несколько нюансов. Во-первых, все эти процедуры вызываются для каждой записи на каждом подписчике. Если на издателе выполнена вставка, обновление или же удаление 1000 строк, соответствующие хранимые процедуры на подписчиках тоже будут вызваны 1000 раз. Во-вторых, вышеперечисленные хранимые процедуры можно изменять, если при этом не изменять сигнатуру их вызова (названия и типы параметров), внутренняя же обработка данных может быть изменена. Например, для нескольких моих проектов было выдвинуто требование – не реплицировать удаления на подписчики. И один из способов это осуществить – просто изменить процедуру удаления вот таким образом:

```
ALTER procedure [dbo].[sp_MSdel_dboSomeNewTable]
  @pkc1 varchar(150)
As – удаления не обрабатываются на этом подписчике
return
```

Единственное, что должно беспокоить при изменении текстов процедур на подписчиках, это пересоздание процедур в случае применения нового снимка. В таком случае лучше создавать завершающие применение снимка сценарии, заменяющие оригинальные процедуры на подписчике измененными. Если в измененной процедуре возникнет ошибка, это приведет к остановке работы агента репликации. Но после исправления ошибки репликация продолжит свою работу с точки сбоя.

Опции доставки

Теперь давайте рассмотрим изменения свойств статей репликации, для чего будем использовать разные опции доставки. В качестве подготовительных мер мною была создана простая публикация репликации транзакций, включающая единственную таблицу ‘SomeNewTable’ и единственного подписчика. Прежде чем я добавил подписчика, был изменен формат доставки выражений вставки на ‘INSERT statement’ как показано далее.

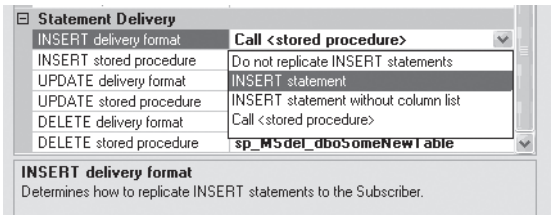


Рис.4.

Что после изменения формата существовало на подписчике:

Рис.5.

Как видите, в приведенном перечне нет процедуры вставки. Если выполнить вставку на издателе то на подписчике, с помощью Profiler, можно отследить такое выражение:

```
exec sp_executesql
  N'insert into [dbo].[SomeNewTable]([EmailAddress],[FirstName],[LastName]) values
(@P1,@P2,@P3)',
  N'@P1 nvarchar(7),@P2 nvarchar(4),@P3 nvarchar(6)',N'5@6.com',N'Andy',N'Warren'
```

Это вполне оптимальное параметризованное выражение вставки, содержащее одну небольшую особенность. Несмотря на то что в моей тестовой таблице столбцы firstname и lastname имеют тип varchar, параметры представлены типом nvarchar – но даже не смотря на это, все работает корректно! Такая техника имеет свои преимущества: мы не должны волноваться о дополнительном коде на подписчике, и поскольку столбцы в выражении вставки перечислены, нет потребности в изменении кода, если порядок столбцов будет изменен. Недостатком же станет невозможность вставки кода для журналирования операций или в целом, невозможность изменения поведения на подписчике. Не имеет значения, каким будет выражение вставки (с полностью определенными столбцами или же только их значениями), синтаксис при отправке изменений подписчикам будет идентичен.

Возвращаясь к опциям вставки, обратите внимание еще на один вариант: «INSERT statement without column list». Эту опцию невозможно установить и сохранить, даже несмотря на возможность выбора ее посредством графического интерфейса не вызывая при этом никаких сообщений об ошибках. Поиск в Интернете не принес каких-либо результатов. В электронной документации по SQL Server (BOL), в главе о процедуре sp_addarticle для команд вставки приведено лишь три доступных опции: не выполнять никаких действий; хранимая процедура;

выражение вставки. Вероятно это лишь недоработка в интерфейсе пользователя.

Последний вариант опции: «Do not replicate...» иногда очень полезен. При его выборе на подписчике не создается хранимая процедура вставки, как и не выполняются вставки посредством sp_executesql. Этот вариант лучше, нежели ухищрения, которые выполнялись в предыдущих версиях, когда на подписчиках изменялись хранимые процедуры вставки, с целью возвращения успешного выполнения, не производя никаких вставок. При старом подходе достигалась эта же цель. Но присутствовала и дополнительная нагрузка, в виде сохранений и распространений записей о вставках, произошедших на издателе. Это не типичный сценарий, но сейчас есть простой и честный способ его выполнения.

Какую же из опций вставки мы должны использовать? Опцией по умолчанию является вызов хранимых процедур, созданных на подписчике. Эта опция имеет два преимущества. Прежде всего, это возможность подстраивать процесс репликации, изменяя процедуры вставки, к тому же, это опция по умолчанию, следовательно, ее проще применять. Преимуществом же опции выполнения выражений вставки является отсутствие необходимости создавать или обновлять процедуры на подписчиках. С точки зрения производительности, эти опции идентичны, поскольку по-прежнему выполняют построчные операции. Опция “Do not replicate...” также исключительно полезна, когда вам необходимо такое поведение, поскольку позволяет избежать значительных нагрузок как, помещение записей в базу распространителя, отправка записей по сети и последующее применение записей на подписчиках.

Ознакомьтесь теперь с опциями удаления.

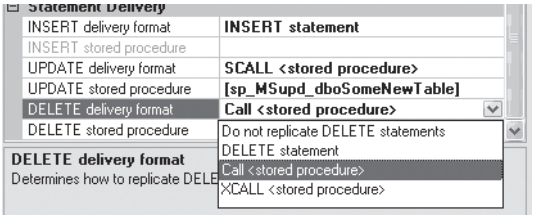


Рис. 6.

Первой представлена опция «Do not replicate DELETE statements», ее действие соответствует названию. Ситуации, когда применяется эта опция, встречаются часто, поскольку иногда подписчики желают сохранять историю данных более длительный срок, нежели издатель. При использовании второй опции на подписчике выполняются выражения T-SQL, подобные этому:

```
exec sp_executesql
  N'delete from [dbo].[SomeNewTable] where [EmailAddress] = @P1'
  , N'@P1 nvarchar(7)',N'2@3.com'
```

Это параметризованные выражения с простым и понятным синтаксисом, результатом чего стоит ожидать хорошую статистику повторного использования планов исполнения для этой конструкции. Перейдем к последней опции, XCALL, и посмотрим, что произойдет при удалении строки:

```
ALTER procedure [dbo].[sp_MSdel_dboSomeNewTable]
  @c1 varchar(150),@c2 varchar(50),@c3 varchar(50)
as
begin
delete [dbo].[SomeNewTable]
where [EmailAddress] = @c1
if @@rowcount = 0
    if @@microsoftversion>0x07320000
        exec sp_MSreplraiserror 20598
end
```

Этот код похож на хранимую процедуру, создаваемую по умолчанию, с разницей лишь в нескольких дополнительных параметрах, даже если они не участвуют в выражении удаления. Предположительно, можно подстраивать процедуру под свои нужды, используя эти дополнительные параметры, но в действительности, это не самая полезная возможность.

Перейдем к опциям обновления: Начнем с обзора всех доступных опций посредством пользовательского интерфейса:

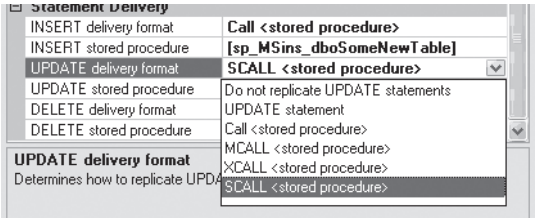


Рис.7.

Первой представлена опция «Do not replicate UPDATE statements». Также как и у рассмотренных аналогичных опций для вставок и удалений, эта опция инструктирует SQL Server полностью игнорировать обновления, избегая лишних расходов на добавление записей в базу распространителя, а затем в базы данных подписчиков. На подписчиках хранимые процедуры обновления не создаются. Далее следует «UPDATE statement». После ее установки и повторного применения снимка можно наблюдать выполнение таких выражений обновления на T-SQL:

```
exec sp_executesql N'update [dbo].[SomeNewTable] set [FirstName] = @P1
where [EmailAddress] = @P2', N'@P1 nvarchar(1),
@P2 nvarchar(14)',N'2',N'test5@test.com'
```

При обновлении на издателе, изменению подвергся лишь столбец FirstName, соответственно только этот столбец был обновлен на подписчике. Поиск строки выполнялся по первичному ключу.

Перейдем к опции «Call». Когда она активна, на подписчиках создается процедура, которая обновляет либо все поля, не входящие в первичный ключ, либо же все поля, включая первичный ключ. Это значит, что все поля будут обновлены, независимо от того, изменились ли их значения на издателе.

```
ALTER procedure [dbo].[sp_MSupd_dboSomeNewTable]
 @c1 varchar(150),@c2 varchar(50),@c3 varchar(50),@pkc1 varchar(150)
as
begin
if not ( @c1 = @pkc1 )
begin
update [dbo].[SomeNewTable] set
 [EmailAddress] = @c1
,[FirstName] = @c2
,[LastName] = @c3
where [EmailAddress] = @pkc1
if @@rowcount = 0
    if @@microsoftversion>0x07320000
        exec sp_MSreplraiserror 20598
end
else
begin
update [dbo].[SomeNewTable] set
 [FirstName] = @c2
,[LastName] = @c3
where [EmailAddress] = @pkc1
if @@rowcount = 0
    if @@microsoftversion>0x07320000
        exec sp_MSreplraiserror 20598
end
end
```

Далее будет рассмотрен XCALL. При ознакомлении со следующей процедурой можно увидеть, что параметрами передаются как оригинальные, так и измененные значения. Аналогично, как и в опции удаления XCALL, это предоставляет возможность изменять логику хранимой процедуры, используя и оригинальные значения, но в реальных ситуациях не так много причин выполнять такие модификации.

```
ALTER procedure [dbo].[sp_MSupd_dboSomeNewTable]
 @c1 varchar(150),@c2 varchar(50),@c3 varchar(50),@c4 varchar(150),@c5
varchar(50),@c6 varchar(50)
as
begin
if not ( @c4 = @c1 )
```

```
begin
update [dbo].[SomeNewTable] set
 [EmailAddress] = @c4
,[FirstName] = @c5
,[LastName] = @c6
where [EmailAddress] = @c1
if @@rowcount = 0
    if @@microsoftversion>0x07320000
        exec sp_MSreplraiserror 20598
end
else
begin
update [dbo].[SomeNewTable] set
 [FirstName] = @c5
,[LastName] = @c6
where [EmailAddress] = @c1
if @@rowcount = 0
    if @@microsoftversion>0x07320000
        exec sp_MSreplraiserror 20598
end
end
```

На очереди остались опции обновления SCALL и MCALL. Интересной особенностью этих опций является то, что если первичный ключ можно обновить, хранимые процедуры будут иметь код, как процедура вставки, представленная в первой части этой серии. Но если будет выбрана таблица, первичный ключ которой не обновляемый (например, поле identity) на подписчике будет создана более простая процедура, представленная далее, для опции SCALL. Второй показана процедура для опции MCALL. Разница между ними только в строке параметров; у SCALL значениями по умолчанию для параметров выступает null, у MCALL – нет.

```
--SCALL синтаксис при не обновляемых первичных ключах
ALTER procedure [dbo].[sp_MSupd_dboEmployees]
 @c1 int = null
 ,@c2 datetime = null
 ,@c3 varchar(50) = null
 ,@c4 varchar(50) = null
 ,@pkc1 int
 ,@bitmap binary(1)
as
begin
update [dbo].[Employees] set
 [DateAdded] = case substring(@bitmap,1,1) & 2 when 2 then @c2 else [DateAdded] end
,[FirstName] = case substring(@bitmap,1,1) & 4 when 4 then @c3 else [FirstName] end
,[LastName] = case substring(@bitmap,1,1) & 8 when 8 then @c4 else [LastName] end
where [EmployeeID] = @pkc1
if @@rowcount = 0
```

```
    if @@microsoftversion>0x07320000
        exec sp_MSreplraiserror 20598
end
--MCALL синтаксис при не обновляемых первичных ключах
ALTER procedure [dbo].[sp_MSupd_dboEmployees]
 @c1 int
 ,@c2 datetime
 ,@c3 varchar(50)
 ,@c4 varchar(50)
 ,@pkc1 int
 ,@bitmap binary(1)
as
begin
update [dbo].[Employees] set
 [DateAdded] = case substring(@bitmap,1,1) & 2 when 2 then @c2 else [DateAdded] end
,[FirstName] = case substring(@bitmap,1,1) & 4 when 4 then @c3 else [FirstName] end
,[LastName] = case substring(@bitmap,1,1) & 8 when 8 then @c4 else [LastName] end
where [EmployeeID] = @pkc1
if @@rowcount = 0
    if @@microsoftversion>0x07320000
        exec sp_MSreplraiserror 20598
end
```

Стоит рассмотреть еще одну опцию обновлений, которая не представлена посредством пользовательского интерфейса. Выполнение sp_scriptdynamicupdateproc сгенерирует хранимую процедуру, формирующую и выполняющую динамический SQL на подписчике. Выполнение этой процедуры лишь сформирует сценарий, запуск его на всех подписчиках должен быть выполнен вручную. Преимуществом этой опции является обновление на подписчике только тех полей, которые были в действительности изменены на издателе, что приведет к увеличению производительности если множество столбцов таблицы входят в индексы, и в это же время станет причиной дополнительных расходов ресурсов на построение и выполнение выражений при каждом запуске. Далее представлен пример созданного на издателе сценария, на основании тестовой таблицы:

```
if object_id(N'[sp_MSupd_dboSomeNewTable]', "P") > 0
    drop proc [sp_MSupd_dboSomeNewTable]
go
if object_id(N'dbo.MSreplication_objects') is not null delete from
dbo.MSreplication_objects where object_name = N'sp_MSupd_dboSomeNewTable'
go
create procedure [sp_MSupd_dboSomeNewTable]
 @c1 varchar(150)
 ,@c2 varchar(50)
 ,@c3 varchar(50)
 ,@pkc1 varchar(150)
 ,@bitmap binary(1)
```

```
as
begin
declare @stmt nvarchar(4000), @spacervar nvarchar(1)
select @spacervar =N''
select @stmt = N'update [dbo].[SomeNewTable] set “
if substring(@bitmap,1,1) & 1 = 1
begin
select @stmt = @stmt + @spacervar + N'[EmailAddress]' + N'=@new1'
select @spacervar = N','
end
if substring(@bitmap,1,1) & 2 = 2
begin
select @stmt = @stmt + @spacervar + N'[FirstName]' + N'=@2'
select @spacervar = N','
end
if substring(@bitmap,1,1) & 4 = 4
begin
select @stmt = @stmt + @spacervar + N'[LastName]' + N'=@3'
select @spacervar = N','
end
select @stmt = @stmt + N'
where [EmailAddress] = @1
“
exec sp_executesql @stmt, N'
 @1 varchar(150)
,@new1 varchar(150)
,@2 varchar(50)
,@3 varchar(50)
“, @pkc1
,@c1
,@c2
,@c3
if @@rowcount = 0
    if @@microsoftversion>0x07320000
        exec sp_MSreplraiserror 20598
end
go
if columnproperty(object_id(N'dbo.MSreplication_objects'), N'article', 'AllowsNull') is not
null
exec (“insert dbo.MSreplication_objects (object_name, publisher, publisher_db,
publication, article, object_type) values (
          + N''sp_MSupd_dboSomeNewTable'' , N''EARLGREY'' , N''ReplTest'' , N''Test'' ,
N''SomeNewTable'' ,''P'')')
```

В следующей таблице приведена обобщенная информация по всем опциям.

	Вставка	Обновление	Удаление
Do not replicate	Вставки не отслеживаются и не передаются	Обновления не отслеживаются и не передаются	Удаления не отслеживаются и не передаются
XCALL	н/д	Используется хранимая процедура, передаются оригинальные и обновленные значения	Используется хранимая процедура, передаются оригинальные и обновленные значения
MCALL	н/д	Используется хранимая процедура, передаются оригинальные и обновленные значения, а также битовая карта измененных полей. Обновления полей происходят либо новыми значениями, либо текущими. Не поддерживаются обновления первичных ключей по identity.	н/д
SCALL	н/д	Используется хранимая процедура, передаются значения для измененных полей, значения по умолчанию для не измененных, а также битовая карта измененных полей. Обновления происходят либо новыми значениями, либо текущими (значениям по умолчанию). Не поддерживаются обновления первичных ключей по identity.	н/д
UPDATE	н/д	Генерируется T-SQL выражение для обновления	н/д
INSERT	Посредством sp_executesql в выражения подставляются названия столбцов и их значения	н/д	н/д
CALL	Используется хранимая процедура, передаются названия столбцов	Используется хранимая процедура, передаются обновленные значения, обновляются все поля	н/д

	Вставка	Обновление	Удаление
sp_script-dynamicup-dateproc	н/д	Используется хранимая процедура, которая формируется и устанавливается на подписчиках вручную, обновляет только измененные поля.	н/д

Выводы

В большинстве случаев значения по умолчанию оправдывают свое существование, и есть немного причин их изменять. Опция «Do not replicate...» очень уместна и должна использоваться в подходящих сценариях, остальные же в особых случаях, но главное – знать о существовании таких опций. Хранимая процедура sp_scriptdynamicupdateproc будет полезна в случае наличия проблем с производительностью на подписчике.

Репликация слиянием

Пакетные обновления в репликации слиянием в SQL Server 2005

По материалам статьи Найджела Манеффа
«Merge Batch Updates in SQL Server 2005»
Перевод Натальи Владимировны Кривонос

Введение

Если вы обновляете существующую установку репликации слиянием в SQL2000 или пишете новое приложение, вы должны знать об изменениях в репликации слиянием, которые могут почти незаметно или, наоборот, весьма серьезно повлиять на ваше приложение, в зависимости от того, как оно написано – это новая особенность, которая называется «пакетные обновления в репликации слиянием». В MSDN2 она отмечена как ***«Изменения, которые могут нарушить работоспособность»*** и эта статья объяснит, что, вероятно, может сломаться, и что мы можем предпринять по этому поводу!

Поведение в SQL 2000

В SQL2000 (и SQL 7.0) все операции update, insert и delete, передаваемые агентом слияния, переносились по одной записи, с использованием репликационных хранимых процедур. Таким образом, если вы делали 20 операций update в таблице, то, когда работал агент слияния, он должен был сделать 20 одиночных вызовов процедур на подписчике.

Новое поведение в SQL 2005

Напротив, в SQL2005 изменение записей может быть выполнено одним пакетом из 20 операций. Для того чтобы обеспечить такое поведение, для каждого действия существует 2 хранимые процедуры. Таким образом, для вставки записей в одну из моих статей я вижу следующее:

```
'dbo.MSmerge_ins_sp_00083C5D3D03432798D51F26F199477C'
'dbo.MSmerge_ins_sp_00083C5D3D03432798D51F26F199477C_batch'
```

Первая процедура – стандартная, а вот вторая (из выделенных курсивом) имеет суффикс «_batch», и это та самая новая процедура, приме-

няемая в данном случае. Если мы посмотрим на эту процедуру, используя следующий запрос:

```
sp_helptext 'dbo.MSmerge_ins_sp_00083C5D3D03432798D51F26F199477C_batch'
```

то увидим, что она ограничивается 100 записями, и все значения GUID посылаются в этой процедуре.

Зачем сделаны такие изменения в SQL2005?

Ответ простой: производительность множественных изменений в одном пакете заметно повышается, хотя потенциально это может несколько ухудшить одновременную работу нескольких соединений. Несколько лет назад я провел несколько экспериментов (совершенно не систематизированных, они проводились слишком давно, и я ничего не документировал), в этих экспериментах я использовал пакетное обновление из клиент-серверного приложения в локальной сети, и в случае однопользовательской работы я получил примерно 30% повышение производительности путем использования пакета из 30 операций вместо одной, и это соотношение сохранялось для таблиц разного размера. Таково преимущество одной большой операции update вместо 30 операций update, по одной на каждую запись. Однако, как это бывает с большинством улучшений, за это приходится платить. В этом случае ваши пользовательские триггеры должны уметь обрабатывать операции update, затрагивающие сразу несколько строк.

Примеры случаев, когда пользовательские триггеры должны быть модифицированы

Почти все триггеры, которые я писал, были созданы с опцией «not for replication», что означает, что они не срабатывают, когда агент слияния синхронизирует изменения. Однако, есть одно исключение, когда триггер создан для маленькой нереплицируемой таблицы, представляющей собой сводную выборку из намного большей реплицируемой таблицы, например, товаров. Этот триггер сознательно создается с опцией «for replication», поскольку нужно, чтобы он срабатывал всегда, включая случаи, когда большая таблица товаров обновляется с удаленного узла. Эта таблица используется, чтобы очень быстро отражать основные сведения о товарах, и, как только пользователь останавливается на одном конкретном товаре, более детальные данные о нем берутся из главной таблицы. Этот триггер – только часть моей системы, где я смог значительно повысить производительность, используя курсоры вместо «классического» решения, использующего работу с множествами. Фактически, в базе

данных размером 10 Гб, решение с курсорами было единственно подходящим.

99.999% изменений, производимых в главной таблице (миллионы строк) — это операции updates/inserts/deletes, затрагивающие одну запись, но если у службы поддержки возникает необходимость скорректировать несколько записей о товарах, они могут обновлять сразу множество записей. Очень редко мы можем изменять тысячи записей, когда выполняем задачи технического обслуживания. Во время тестирования я обнаружил, что триггер, использующий курсор, работает хорошо, пока количество обрабатываемых записей не превышает 20, но его производительность резко падает, когда количество записей доходит до 1000. Так что я переделал его так, что он не пытался изменить сводную таблицу, если в главной таблице происходило больше 50 изменений в одном пакете (за одну операцию update), а вместо этого триггер делал запись в таблицу errorlog. Таблица errorlog автоматически очищалась, при этом по электронной почте посылалось сообщение о требуемых действиях, потом триггер завершался, не выполняя никаких действий (вследствие чего сводная таблица оставалась несогласованной с главной). Опыт показывает, что в этом случае было бы дешевле запускать код, который бы пересоздавал сводную таблицу целиком; это занимало бы около минуты, и выполнялось бы во время наименьшей загрузки.

Гуру среди вас могли бы посмеяться и предложить использовать материализованные представления, которые были бы намного лучшим решением. Вы были бы правы, но код, который я тестировал на SQL 2000 SP2, оказался слишком сложным для репликации слиянием и создавал странные конфликты, которые удаляли записи из таблиц! (Я обнаружил это, когда сделал первое настоящее обновление после того, как весь код целиком был написан – помните: всегда используйте полную настройку репликации с издателем и подписчиком для окончательного тестирования!!!), и потом столкнулся с катастрофической проблемой в SP3 (http://support.microsoft.com/kb/816780), которая надолго заставила меня отказаться от материализованных представлений. Этот код отлично работает на SQL2005, конечно, с использованием курсоров.

Тем временем я узнал о потенциально катастрофической для меня возможности при инсталляции SQL2005 – что количество записей при пакетном обновлении может оказаться слишком большим для моего бедного триггера. В конце концов, триггер для пакетного обновления может получать для обновления 100 записей за один раз, а мой триггер ограничен 50. Это будет приводить к постоянным ошибкам, поскольку мой пользовательский триггер будет перегружен, или даже хуже того, количество записей в пакете будет сильно варьироваться в зависимости от размера поля или количества полей в таблице – и это приведет к тому, что я, не подозревая ничего плохого, сделаю однажды безобидное изменение поля, и это приведет к тому, что мой триггер вообще перестанет срабатывать при пакетных обновлениях. Я должен был исследовать этот

вопрос. Данной информации не было в электронной документации по SQL Server, так что я создал тестовую среду и нашел все ответы сам. Вы можете посчитать эту информацию полезной, так что вот она:

Тестирование

Базовая таблица для тестирования выглядит следующим образом:

```
CREATE TABLE mergebatchtest(
   pk int NOT NULL,
   col1 char(10) NOT NULL,
   rowguid uniqueidentifier ROWGUIDCOL NOT NULL
   CONSTRAINT DF_mergetriggertest_guid DEFAULT (newid()) FOR rowguid)
```

Я провел 3 теста:

1. Если меняется размер поля, как это влияет на количество пакетов?
2. Если у нас становится больше полей, как это влияет на количество пакетов?
3. Сколько нужно накопить изменений, чтобы для них был создан новый пакет?

Результаты

1. Если меняется размер поля, как это влияет на количество пакетов? Я обнаружил, что типы и количество полей не влияют на количество пакетов.
2. Если у нас становится больше полей, как это влияет на количество пакетов? Я добавлял поле char(10) снова и снова и следил в триггере за количеством изменений, приходящих в одном пакете. Интересно то, что количество изменений в одном пакете очень сильно варьировалось в зависимости от количества полей в таблице. Смотрите приложение для получения точной количественной информации.
3. Сколько нужно накопить изменений, чтобы для них был создан новый пакет? Ответ оказался очень простым. Функциональность пакетных изменений осуществляет вызов пакетной хранимой процедуры, как только накапливается более одной операции update/insert/delete для таблицы (или статьи по терминологии Майкрософт). На самом деле с операциями удаления пакетная обработка включалась, даже если было только одно удаление в моей среде (затраты на работу пакетной процедуры для обработки удалений были достаточно низки, чтобы запускать ее всегда). Для операций update (и insert) «одиночная» хранимая процедура запускалась всегда, когда было только одно обновление, и «пакетная» версия запускалась, как только накапливалось 2 и больше изменений. Довольно разумно.

[Если вы дочитали до этого места, то знайте, что момент истины настал, когда я обнаружил, что SQL2005 не перегружает мой триггер!!! Главная таблица содержала 44 поля, и при этом количество изменений в одном пакете ограничивалось примерно 20, так что я был спасен и должен был бы приложить немалые усилия для сокращения количества полей до 15, чтобы создать проблемную ситуацию в моей системе работы с товарами!]

Заключение

Вместе со значительным увеличением производительности, пакетные обновления при репликации слиянием должны приниматься во внимание при разработке баз данных и приложений. Было бы удобно, если бы компания Microsoft предоставила параметр командной строки, позволяющий отключать функциональность пакетных обновлений приложениям, которые не могут управлять триггерами с многострочной обработкой по тем или иным причинам. Я подозреваю, что такой параметр может существовать, но только для внутренней отладки или тестирования производительности, но если это и так, то он недокументирован и, возможно, таким и останется.

Приложение

Следующая таблица показывает соотношение между количеством полей в таблице и количеством изменений в пакете.

Количество полей в таблице	Количество изменений в пакете
3	100
4	92
5	85
6	78
7	73
12	53
17	46
22	37
27	32
32	27
37	24
42	21
47	20
52	18
62	17
72	14

Количество полей в таблице	Количество изменений в пакете
82	12
92	10
127	7
152	6
202	4

Поскольку эти цифры неофициальны теоретически Майкрософт может изменять триггеры с выпуском новых сервис-паков, хотя я сомневаюсь, что они будут делать это, пока пользователи не столкнутся в этих триггерах с необычными проблемами, что маловероятно.

(под технической редакцией Пола Ибисона)

Изменения в репликации слиянием: Динамические фильтры и однонаправленная репликация

По материалам статьи Пола Ибизона (Paul Ibison)
«Merge Changes: 1 – Dynamic Filters and Unidirectional Merge»
Перевод Натальи Владимировны Кривонос

Введение

Эта статья написана, чтобы окинуть взглядом новую функциональность в репликации слиянием, используя SQL Server 2005 Beta CTP от апреля 2005. В частности, меня интересовало, как реализованы две различные техники: динамические фильтры и однонаправленная репликация слиянием. Почему именно они? Ну, потому что я всегда считал, что в SQL 2000 реализация была скорее неудовлетворительной... Она включает в себя ручное редактирование текста в этапах заданий, что является редким исключением в администрировании репликации. Это делает обслуживание более сложным, и ранее я делал запрос на добавление функциональности, которая бы позволяла добавлять в профиль агента слияния параметры, которые избавляли бы от необходимости заниматься низкоуровневым редактированием. Так что мне интересно было посмотреть, была ли эта функциональность изменена для того, чтобы сделать конфигурацию более сложной, а нашу жизнь более легкой.

Базовая информация

Если читатель не совсем знаком с этими понятиями, то ниже приводятся (очень выборочно) краткие сведения.

Динамические фильтры

Фильтры могут быть статическими и динамическими. В случае статических фильтров все подписчики получают одни и те же данные. При-

мер статического фильтра – *Country = 'UK'*. В этом случае данные в таблице фильтруются во время синхронизации так, что все подписчики получают только данные по UK. Динамические фильтры работают так, что каждый подписчик получает (потенциально) отдельную горизонтальную часть данных. Подписчик идентифицируется в условии фильтра с помощью динамической функции. Пример динамического фильтра – *Country = HOST_NAME()*, где функция HOST_NAME() возвращает NETBIOS-имя компьютера подписчика (для подписки по запросу). Для того чтобы не переименовывать каждый компьютер подписчика так, чтобы его NETBIOS-имя соответствовало названию страны, общим решением является использование параметра -HOSTNAME в описании этапа задания агента слияния, который перекрывает значение, возвращаемое динамической функцией. Например, так: *-HOSTNAME Wales*. Другим, менее часто встречающимся решением, является написание пользовательской функции, но давайте пока забудем о нем. Подведем итог, параметр -HOSTNAME недоступен при настройке подписки или через профиль агента, и в большинстве случаев администраторы вынуждены прибегать к редактированию задания агента слияния, чтобы это заработало.

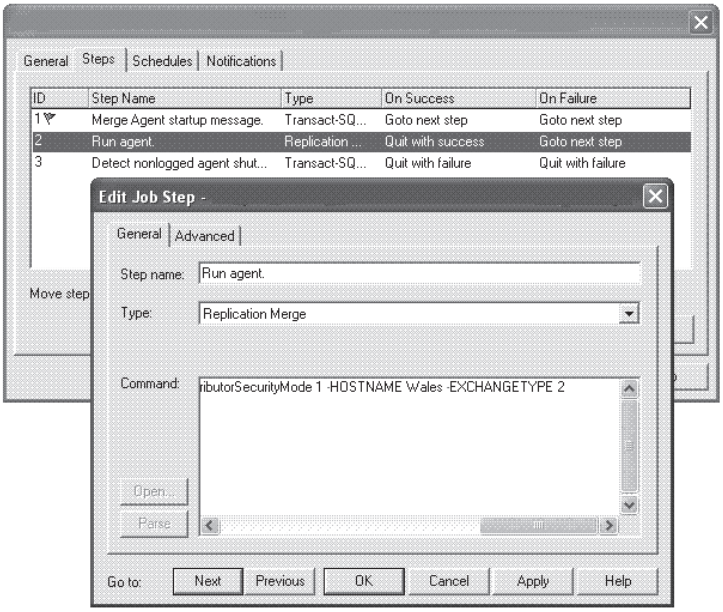
Однонаправленное слияние

Значение -EXCHANGETYPE определяет направление передачи изменений в репликации слиянием. И опять это выполняется путем ручного редактирования этапа задания агента слияния, этот параметр задается как -EXCHANGETYPE 1|2|3 в текстовом виде. Значения 1|2|3 (символ '|' означает 'или') объясняются ниже:

ВЫГРУЗКА	1	Только изменения Подписчика пересылаются на Издателя.
ЗАГРУЗКА	2	Только изменения Издателя пересылаются на Подписчика.
ДВУНАПРАВЛЕННАЯ	3	Передаются все изменения между Издателем и Подписчиком (режим по умолчанию).

По умолчанию поток изменений является двунаправленным (-EXCHANGETYPE = 3). Изменение значения на 2 означает, что изменения в реплицируемой статье на подписчике не запрещены, они записываются в таблицы метаданных репликации слиянием с помощью репликационных триггеров и впоследствии отфильтровываются, когда агент слияния синхронизирует данные. Это означает, что может накапливаться большое количество ненужных метаданных, что замедляет изменения данных и синхронизацию.

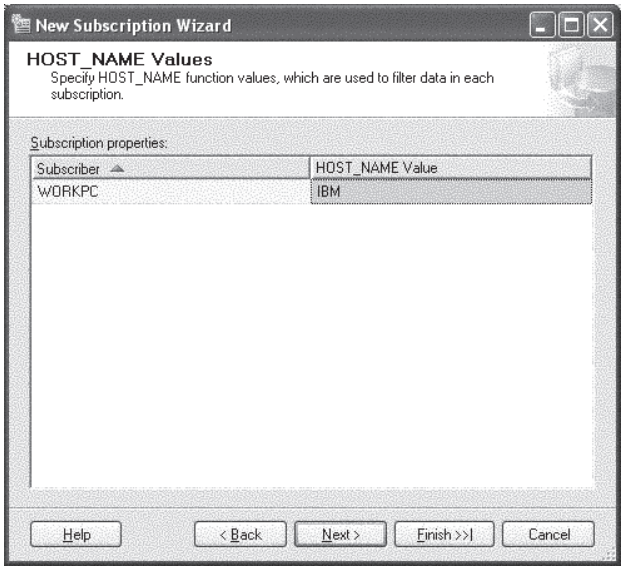
Так что, если мы комбинируем динамические фильтры и однонаправленные изменения, то этап задания агента слияния должен выглядеть следующим образом, чтобы загружать данные только по Уэльсу:



Что изменилось в SQL Server 2005?

Динамические фильтры

При добавлении подписчика вы видите форму, позволяющую определить (жестко задать) значение для функции HOST_NAME(), которое перекрывает NETBIOS-имя.



В виде строки кода на T-SQL это будет так:

```
sp_addmergesubscription @hostname = 'hostname'
```

и называется «определение секции». Так что вам больше не надо редактировать этапы заданий.

Как и в SQL Server 2000, это значение перекрывает значение, возвращаемое функцией HOST_NAME(). Входной параметр HOSTNAME, полученный из пользовательского интерфейса, используется для установки свойств в таблице MSsubscription_properties, и затем агент слияния может читать это свойство из указанной таблицы во время последующих сеансов синхронизации. Это справедливо для подписок по запросу. Однако в случае принудительных подписок не существует таблицы, которая управляла бы такими свойствами на издателе, так что строка «-HOSTNAME host_name» записывается в командную строку задания на распространителе. Поэтому, если вы забыли присвоить это значение во время создания подписки, его возможно задать для принудительных подписок, пользуясь той же методологией, что в SQL Server 2000, и вручную вписать строку «-HOSTNAME host_name» в этап задания слияния. Параметр командной строки –HOSTNAME, если он задан (администратором или с помощью пользовательского интерфейса), используется и всегда перекрывает другие источники этого значения. Например, свойство HOSTNAME в таблице MSsubscription_properties может быть перекрыто указанием значения –HOSTNAME в командной строке.

Однонаправленное слияние

При добавлении статьи в публикацию, существует настройка, позволяющая задать параметр «subscriber_upload_options»:

```
sp_addmergearticle @subscriber_upload_options= subscriber_upload_options
```

Этот параметр определяет ограничения на изменения записей, сделанные на Подписчике (с клиентской подпиской). Параметр «subscriber_upload_options» имеет тип tinyint и может принимать одно из следующих значений.

0 Нет ограничений. Изменения, сделанные на Подписчике, передаются на Издателя.

1 Изменения разрешены на Подписчике, но они не передаются на Издателя.

2 Изменения не разрешены на Подписчике.

Так как же это теперь реализовано?

Значение 0 определяется как «без ограничений», «двунаправленное слияние» и является значением по умолчанию. Больше всего это похоже на использование «-EXCHANGETYPE = 3» в SQL Server 2000. Обычно при этом на подписчике создается 3 триггера, чтобы протоколировать все изменения данных в таблицах метаданных:

MSMerge_ins_...MSMerge_upd_...MSMerge_del_...

Значение 1 определяется как «только выгружать, но изменения на подписчике разрешены». Это эквивалентно «-EXCHANGETYPE = 2», но в этом случае триггеры на подписчике не создаются. Таким образом, хотя оба этих метода логически эквивалентны, в SQL Server 2005 реализация существенно улучшена :). Нет срабатывания триггеров для протоколирования метаданных на подписчике, что ускоряет как изменения на подписчике, так и последующую синхронизацию.

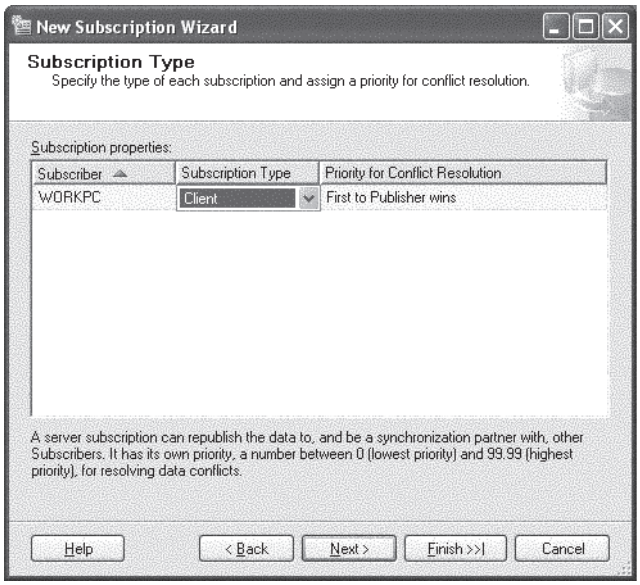
Значение 2 запрещает все изменения на подписчике. В этом случае создается специальный триггер - MSmerge_downloadonly_.... – который будет откатывать любую попытку изменения данных на подписчике. Эта возможность является новой для SQL Server 2005 и не имеет эквивалента в SQL Server 2000. Если пользователь сделает попытку изменения данных, он получит следующее сообщение:

```
Msg 20063, Level 16, State 1, Line 1
Table '(null)' into which you are trying to insert, update, or delete data has been marked
as read-only. Only the merge process can perform these operations.
```

Параметр «-EXCHANGETYPE [1|2|3]» для задания агента слияния попрежнему доступен, даже если кажется, что он в значительной степени перекрывается новой функциональностью. Этот параметр до сих пор работает с двунаправленной репликацией (@subscriber_upload_options=*2)*, но для повышения эффективности я бы не рекомендовал использовать -EXCHANGETYPE как альтернативу соответствующему значению @subscriber_upload_options. Возможно, стоит сделать исключение в случае -EXCHANGETYPE = 1, поскольку не существует аналогичного значения для @subscriber_upload_options.

Примечание. При указании параметра EXCHANGETYPE пропускается фаза загрузки или выгрузки агента слияния, и это оказывает влияние на *все* статьи в публикации, в то время как @subscriber_upload_options в SQL 2005 указывается на уровне *статьи*. Так что теперь вы можете иметь подмножество статей в публикации, данные в которых будут только выгружаться. Это типичный сценарий для клиента, когда некоторые статьи используются только для просмотра (такие, как каталог цен на продукты), но существуют и другие статьи, репликация данных которых должна быть двунаправленной.

И последний момент – для того, чтобы получить преимущества этой функциональности (@*subscriber_upload_options)*, подписка должна быть помечена как «Клиентская», а не «Серверная», что является значением по умолчанию (см. ниже). Если вы оставите значение «Серверная», то подписчик всегда будет создаваться как двунаправленный.



Заключение

Мы рассмотрели замену для параметров репликации слиянием -EXCHANGETYPE и -HOSTNAME в SQL Server 2005. Новая функциональность – это большой шаг вперед, сделанный для явного указания свойств как публикаций, так и подписок. Это не реализовано как изменение дополнительного параметра профиля агента слияния, как я надеялся (и посылал запрос), но результат достигается тот же самый – больше не требуется ручное редактирование заданий репликации, чтобы все работало так, как надо. В случае замены -EXCHANGETYPE на *@subscriber_upload_options* мы видим значительное увеличение производительности, и к тому же более тонкое управление, которое осуществляется на уровне статьи, а не целой подписки.

Веб-синхронизация в SQL 2005 для репликации слиянием

По материалам статьи Хилари Котер (Hillary Cotter)
«SQL 2005 Web Synchronization for Merge Replication»
Перевод Владислава Александровича Щербинина

Веб-синхронизация в SQL 2005 позволяет выполнять безопасную синхронизацию подписок в репликации слиянием. Потребность в такой возможности была очень нужна много лет, и наконец-то она была разработана компанией Microsoft. Однако эту возможность сложно настраивать, так что Hilary Cotter по шагам описывает процесс установки. До выхода SQL Server 2005, при необходимости репликации через Internet, сетевому администратору потребовалось бы настроить виртуальную частную сеть - VPN или открыть на межсетевом экране порт 1433 (порт tcp/ip, используемый SQL Server по умолчанию) и порт 21 (порт tcp/ip, используемый протоколом FTP) для входящих соединений.

VPN не популярны по следующим причинам:

- Эта технология обычно медленная.
- Сложно настраивать и обеспечивать безопасность.

Большинство системных администраторов не расположены открывать порты 1433 и 21, потому что в результате SQL Server будет выставлен в Internet и может быть найден хакерами, использующими сканеры портов.

Использование FTP по определению не безопасно, так как:

- Если не используется анонимная аутентификация, имя учетной записи и пароль будут передаваться открытым текстом.
- Если используется анонимная аутентификация, кто угодно сможет получать доступ к ftp-серверу и загружать его содержимое.

К счастью, Microsoft SQL 2005 предоставляет безопасный способ выполнения синхронизации для репликации слиянием через Internet. Веб-синхронизация безопасна, так как:

- Для входящих соединений на межсетевом экране открыт только порт 443. Порт 443 используется для осуществления безопасного взаимодействия с веб-серверами (https), и большинство сетевых админист-

раторов открывают этот порт. Вся информация, передаваемая с использованием порта 443, включая имена учетных записей, пароли и данные синхронизации, шифруется.

- Подписчики в репликации слиянием для синхронизации соединяются с SQL Server посредством веб-сервера — SQL Server вообще не выставляется в Internet.

Веб-синхронизация доступна только в SQL Server 2005 Enterprise Edition, работающем на Windows 200x Enterprise Edition с сервером IIS. Несмотря на то, что возможно настроить веб-синхронизацию на SQL Server 2005 Developer Edition, работающем на Windows XP Professional для целей тестирования, с точки зрения лицензирования, такая конфигурация может использоваться только для тестирования.

Веб-синхронизация будет хорошо знакома администраторам баз данных и SQL-разработчикам, если они работали с репликацией с Pocket PC и SQL CE. Фактически, Microsoft взяла веб-синхронизацию из SQL CE, и сделала ее безопасной, разрешив взаимодействие только по https. Веб-синхронизацию можно использовать не только с SQL CE, но также и с SQL Mobile, SQL Express всеми вариантами SQL 2005.

На рис. 1 показано, как может выглядеть топология веб-синхронизации.

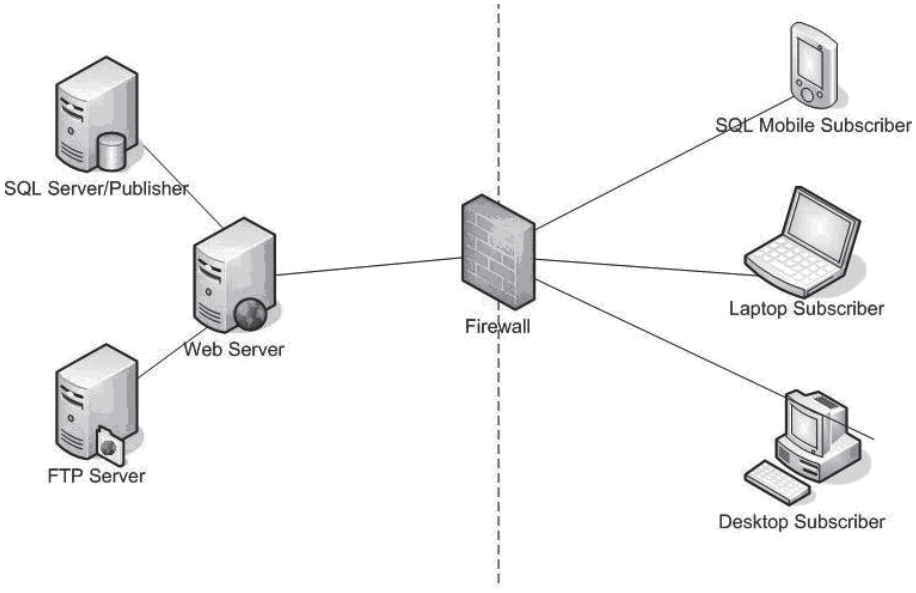


Рис. 1. Типовая топология для среды веб-синхронизации

Теперь, когда понятно, как выглядит топология, рассмотрим шаги, необходимые для ее настройки.

Настройка веб-сервера для веб-синхронизации

Первое, что надо сделать, это настроить веб-сервер для веб-синхронизации. Для этого необходимо получить сертификат. Сертификат можно создать самостоятельно, используя Certificate Services, поставляемые с Windows 200x Enterprise Edition, или приобрести у провайдеров сертификатов, таких как Thwarte, Verisign, и так далее.

Чтобы установить сертификат на веб-сервер, откройте оснастку Internet Information Services, раскройте IIS, раскройте узел для сервера/рабочей станции, найдите веб-сайт, который будет выступать хостом для процесса синхронизации, нажмите на нем правую кнопку, выберите Properties, и перейдите на вкладку Directory Security. Нажмите кнопку Server Certificate, чтобы установить сертификат. После того, как это будет выполнено, откройте браузер и введите **https://localhost**/, чтобы удостовериться, что SSL работает. Также попробуйте указать полное квалифицированное имя домена, которое подписчики будут использоваться для соединения с веб-сервером при выполнении веб-синхронизации. Вам будет предложено принять сертификат, если установка SSL была выполнена корректно.

Настройка сервера-издателя для веб-синхронизации

Для настройки сервера-издателя для веб-синхронизации, подключитесь к издателю с помощью SQL Server Management Studio (SSMS), раскройте папку Replication, раскройте папку Local Publications, нажмите правую кнопку на публикации репликации слиянием, для которой требуется включить веб-синхронизацию.

Будет показано окно мастера Welcome to Configure Web Synchronization Wizard. Нажмите Next.

В диалоговом окне Subscriber Type выберите тип подписчиков, которые у вас будут. Существует два варианта: подписчики SQL Server 2005 (Enterprise, Standard, Workgroup, Developer и Express) и клиенты SQL Server 2005 Mobile. Выберите необходимый вариант и нажмите Next.

Вам будет предложено ввести имя веб-сервера, выступающего в роли хоста для процесса веб-синхронизации. По соображениям безопасности, этот сервер должен отличаться от сервера-издателя. Выберите Create A New Virtual Directory или Configure An Existing Virtual Directory. После выбора веб-сервера, вы можете выбрать существующие виртуальные каталоги, или веб-сайт, на котором хотите создать новый виртуальный каталог.

Примечание. *Виртуальный каталог (virtual directory)* это путь, доступный через браузер (https:// www.MyServerName.com/ MyVirtualDirectory). Этот путь называется виртуальным каталогом, потому что виртуальное имя может отличаться от физического каталога, на который оно отображается (MyVirtualDirectory может отображаться на d:\ReplicationFolder), и физический путь от корня веб-сервера тоже может отличаться (по умолчанию c:\inetpub\wwwroot).

Если вы выберите создание нового виртуального каталога, будет предложено создать новую физическую папку (если она еще не существует). Также будет предложено подтвердить копирование расширения ISAPI, которое будет обрабатывать запросы веб-синхронизации. Выберите Yes для обоих предложений.

Затем вы увидите диалоговое окно для Authentication Access. Microsoft рекомендует использовать Basic Authentication. В случае домена, введите имя домена, как оно выглядит в сертификате (*MyDomain*.com); в случае зон (realm), введите полное квалифицированное имя домена, как оно выглядит в сертификате (*Server.MyDomain*.com). Если вы забыли эти имена, подключитесь к веб-серверу, нажмите правую кнопку на веб-сайте, выберите Properties и Directory Security, и затем нажмите View Certificate.

После нажатия кнопки Next, вы увидите диалоговое окно Directory Access. Выберите учетную запись или группу, которая будет использоваться для подключения к общей папке с моментальным снимком.

Примечание. Для упрощения управления разрешениями, используйте группы, а не индивидуальные учетные записи, потому что при использовании групп разрешения надо будет настраивать только один раз.

Эта группа должна иметь разрешения на доступ к моментальному снимку, который может находиться на издателе, ftp-сервере или файловом сервере. По умолчанию, общая папка моментального снимка имеет разрешение на чтение для группы Everyone (Все). При настройке совместного доступа к общей папке моментального снимка убедитесь, что у группы, которую вы используете, есть разрешения на чтение (read) и просмотр содержимого папки (list folder contents) для общей папки моментального снимка. Если пристальней рассмотреть это диалоговое окно, вы увидите, что в нем есть флажок, установленный по умолчанию: Grant The Above Users Permission To Access The UNC Snapshot Share. Если вы

используете FTP-сервер, убедитесь, что используется анонимная аутентификация, или, что у используемой учетной записи есть разрешения read и list folder contents. Нажмите кнопку Add в этом диалоговом окне и найдите группу.

Нажмите Next, чтобы перейти к диалоговому окну Snapshot Share Access. Введите название общей папки для моментального снимка в формате UNC: *MyServerName**ShareName*. Общая папка должна быть создана заранее. Нажмите Next. Если публикация еще не настроена на использование этой общей папки в качестве папки моментального снимка, будет сообщено, что общая папка пуста. Убедитесь, что именно эту общую папку вы хотите использовать и нажмите Next, чтобы перейти к диалоговому окну мастера Complete The Wizard. Проверьте выбранные вами настройки и нажмите Finish.

После того, как мастер Web Configuration Wizard завершит работу, вы получите отчет об успешных/неуспешных действиях, чтобы можно было определить, какой из компонентов отработал неудачно, и повторно запустить мастер для устранения проблем.

Настройка публикации для веб-синхронизации

Вот так так! После всех этих действий, вы, наверное, рады узнать, что настройка публикации для веб-синхронизации состоит из двух шагов. Подключитесь к SQL Server, используя SSMS, раскройте папку Replication, раскройте Local Publication, нажмите правую кнопку на публикации для репликации слиянием, и выберите Properties. Перейдите на вкладку FTP Snapshot and Internet, и установите флажок Allow Subscribers To Synchronize By Connecting To A Web Server. В самом низу диалогового окна введите путь к веб-серверу и его виртуальный каталог. Путь должен выглядеть подобно следующему: https://*MyServerName.MyDomainName*.com/*MyVirtualDirectory*

Это диалоговое окно показано на рис. 2.

Проверьте настройку публикации, перейдя в Internet Explorer по следующему пути:

https://MyServerName.MyDomainName.com/MyVirtualDirectory/replisapi.dll?diag

Веб-сервер вернет статусную страницу, которая полезна для диагностики различных ошибок настройки.

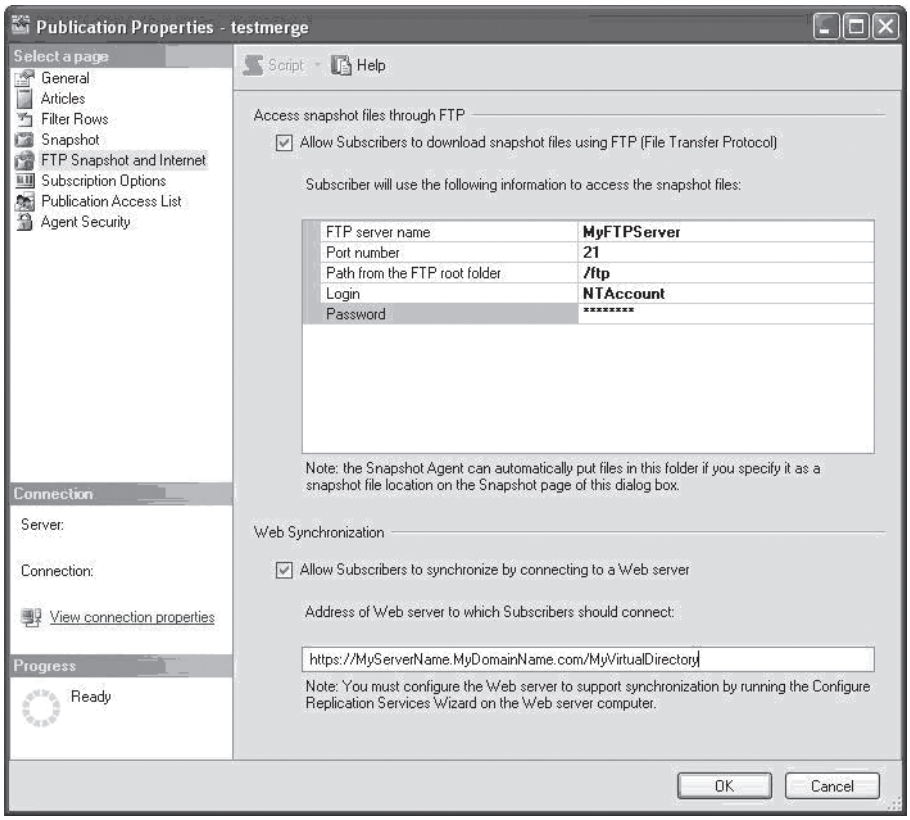


Рис. 2. Диалоговое окно Publication Properties, с выбранной вкладкой FTP Snapshot and Internet

Настройка локальных подписчиков для веб-синхронизации

Если вы можете зарегистрировать подписчиков и издателя в SSMS, все что вам надо будет сделать, это создать новую подписку «по запросу» (pull subscription) на подписчике. Подключитесь к подписчику, раскройте папку Replication, нажмите правую кнопку на папке Local Subscriptions, и выберите New Subscription.

Будет показано диалоговое окно мастера New Subscription Wizard. Нажмите Next. В выпадающем списке Publisher, выберите издателя, раскройте базу данных публикации, выберите публикацию и нажмите Next.

Если издатель здесь не появляется, выберите в выпадающем списке Find SQL Server Publisher и попробуйте к нему подключиться. Если издателя в списке нет, добавьте его как псевдоним, используя SQL Server

Configuration Manager (эта программа находится в Start, All Programs, Microsoft SQL Server 2005, Configuration Tools). Раскройте SQL Server Native Client Configuration и нажмите правую кнопку на узле псевдонимов, чтобы создать новый псевдоним для издателя — используйте TCP/IP в качестве протокола, и имя сервера издателя в качестве псевдонима. Имя сервера издателя можно определить, выполнив команду select @@servername на издателе. В качестве серера можно ввести его IP-адрес. Существует большая вероятность того, что, если подписчик находится в домене, отличном от издателя, или отделен от издателя межсетевым экраном, другой локальной сетью, или глобальной сетью, вам придется следовать инструкциям в разделе «Настройка удаленных подписчиков для веб-синхронизации».

В диалоговом окне Merge Agent Location, выберите Run Each Agent At Its Subscriber (Pull Subscriptions) и нажмите Next. В диалоговом окне Subscribers, выберите подписчика и базу данных, в которой будет находиться подписка. Здесь можно выбрать несколько подписчиков, каждый со своей базой данных подписки. Нажмите Next.

В диалоговом окне Merge Agent security вы можете выбрать учетные записи, которые будут использоваться для подключения к издателю и подписчику. Разрешены два варианта:

- Specify The Domain Or Machine Account Under Which The Merge Agent Process Will Run When Synchronizing This Subscription
- Connect To The Subscriber

В первом варианте учтеная запись издателя должна быть в PAL (Publication Access List, список доступа к публикации), иметь разрешения на чтение и просмотр содержимого папки в общей папке моментального снимка, и также учетная запись должна быть включена в роль db_owner в базе данных распространителя (distribution database).

Это учетная запись, которая используется для подключения и аутентификации на веб-сервере, а затем – для подключения к базе данных и синхронизации. Это должна быть учетная запись из домена издателя или учетная запись на локальной машине, и в диалоговом окне ее надо вводить в формате ИмяДомена\ИмяУчетнойЗаписи.

Учетная запись подписчика должна находиться в домене подписчика или на локальной машине, и так же должна быть включена в роль db_owner в базе данных подписки.

Нажмите Next. Для задачи Synchronization Schedule, выберите расписание. Нажмите Next. В диалоговом окне Initialize Subscription, выберите Initialize Immediately. Нажмите Next. В диалоговом окне Web Synchronization, выберите Use Web Synchronization (см. рис. 3).

Нажмите Next. В этом диалоговом окне для Subscription Type, выберите Client в диалоговом окне Subscription Type.

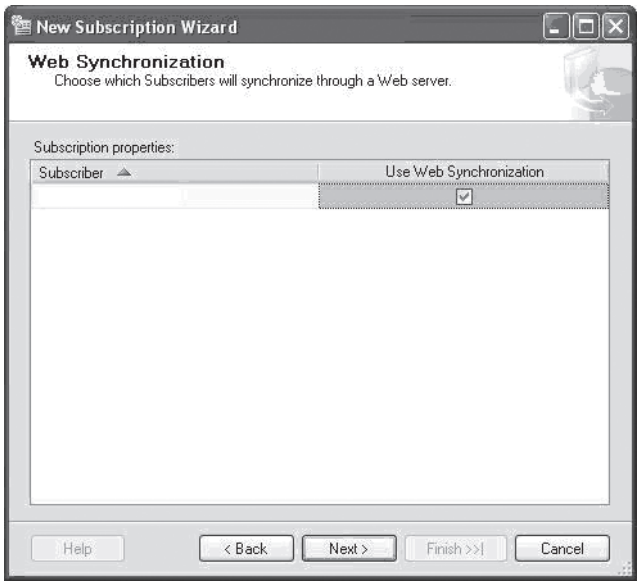


Рис. 3. Включение веб-синхронизации для подписки

В диалоговом окне Wizard Actions, оставьте настройки по умолчанию, и нажмите Next. Проверьте выбранные настройки в диалоговом окне Complete The Wizard, и нажмите Finish, чтобы использовать их. Будет создана подписка.

Настройка удаленных подписчиков для веб-синхронизации

Если подписчик находится в другом домене или между издателем и подписчиком находится межсетевой экран, вы не сможете зарегистрировать издателя в SSMS, так что вы не сможете создать подписку с помощью мастера. Единственный способ создать подписку состоит в использовании SQL RMO, элементов управления ActiveX, или хранимые процедуры для работы с репликацией.

Заключение

На этом завершим руководство по использованию веб-синхронизации. Самая сложная часть процесса настройки – это настройка сертификата, и корректная настройка безопасности. Веб-синхронизация предоставляет высокопроизводительный, безопасный мехнизм репликации по Internet.

Зеркальное отображение базы данных и репликация слиянием

По материалам статьи Ларри Чеснута (Larry Chesnut) «DB Mirroring and Merge Replication»

Перевод Яна Дмитриевича Либермана

На прошлой неделе, когда я проводил очередное занятие, мне задали несколько интересных вопросов, на которые я не сразу нашелся, что ответить. Поэтому мы договорились, что я опубликую вопросы и ответы на них в своем блоге. Один из вопросов был про зеркальное отображение базы данных и репликацию слиянием, где на подписчиках установлен SQL Server 2000. Если произойдет переключение ролей, то будут ли подписчики автоматически переключены на резервный сервер? Будет ли достаточным установить на подписчиках SQL Server 2005 SNAC (SQL Native Access Components) или потребуется установить MDAC 2.9 в полном объеме? Кроме того, так как репликация слиянием подразумевает, что подписчики могут отключаться от сети и выключать свои ноутбуки, а, следовательно, полагаться исключительно на кэш MDAC нельзя, как прописать на подписчиках данные резервного сервера?

Для того чтобы сформулировать вопрос более точно, нужно определить для какой базы данных настроено зеркальное отображение – для распространителя или для издателя. В сценарии, который меня интересует – это издатель.

Ответ на сегодняшний день звучит так: зеркальное отображение баз данных этот сценарий не поддерживает[1]. Как общее правило, зеркальное отображение баз данных не поддерживает объекты уровня сервера, такие как задания (job) и имена входа (login). Также, зеркальное отображение баз данных не обновляет метаданные репликации и не меняет имя сервера после переключения на резервный сервер. Нельзя сказать, что все это нельзя преодолеть, например, реализовав реакцию на соответ-

[1] Статья была написана на основе версии SQL Server 2005 Beta2. На момент подготовки книги (SQL Server 2005 SP2) при использовании зеркального отображения баз данных поддерживается переход репликации на зеркальную базу данных (при сбое основной базы данных) только для баз данных публикации. Переход на резервную базу данных для базы данных распространителя и базы данных подписки не поддерживается. Подробнее см. «Электронная документация по SQL Server 2005« статья «Репликация и зеркальное отображение базы данных».

ствующее уведомление о событии (event notification). Но в рамках этого блога я хотел бы обратить ваше внимание на то, что эта работа не делается для вас автоматически. Вы должны сами решить насколько эта функциональность подходит для вашей системы и готовы ли вы взять на себя дополнительную работу по ее доработке.

Зеркальное отображение базы данных предназначено для обеспечения высокого уровня доступности базы данных. Если вам нужно добиться высокого уровня доступности на уровне сервера, то вы должны рассмотреть возможность внедрения решения на базе отказоустойчивых кластеров (failover clustering).

Есть несколько важных моментов, которые нужно помнить, если вы собираетесь использовать инструменты обеспечения высокого уровня доступности на уровне базы данных (например, зеркальное отображение баз данных). Во-первых, те имена входа (login), которые нужны для базы данных, должны быть созданы как на основном, так и на резервном сервере, причем их SID должны совпадать. Иначе после переключения на резервный сервер, пользователи в базе данных утратят связь с именем входа (login). Во-вторых, не забудьте, что задания (job), созданные на основном сервере, также должны быть созданы и на резервном сервере. Но как создать задания, когда зеркальная база данных недоступна? Кроме того, после создания, эти задания должны быть отключены и оставаться в таком состоянии вплоть до момента, когда произойдет переключение на резервный сервер. Затем задания должны быть включены обратно. Но как в нужный момент автоматически включить все эти задания? Подразумевается, что вы сами должны запрограммировать соответствующую реакцию на событие переключения на резервный сервер. Звучит как потенциальная работа для Service Broker. И конечно, как я писал раньше, зеркальное отображение баз данных и регулярные (например, по расписанию) BCP/Bulk-вставки, особенно если речь идет о действительно больших объемах данных, плохо сочетаются вместе.

Основываясь на вышесказанном, вероятно, стоит рассматривать зеркальное отображение баз данных как способ обеспечить работу системы до тех пор, пока основной сервер не будет введен в эксплуатацию. Как только это произойдет, вы сможете переключиться обратно на него.

Я хочу обратить ваше внимание на то, что между зеркальным отображением баз данных и отказоустойчивыми кластерами есть существенная разница. В случае зеркального отображения баз данных, хранилище данных (data storage) не является общим. Использование отказоустойчивых кластеров помогает, в случае, например, если процессор или память выйдет из строя или произойдет серьезная ошибка в операционной системе. Но как часто вы видите на своей системе «синий экран»? Вероятно, не часто. Зеркальное отображение баз данных может защитить вас еще и от неисправности дисковой подсистемы, так как в этом случае поддерживается полная отдельная копия данных. Мне видится, что вероятность отказа диска или дискового контроллера больше, чем вероят-

ность отказа какой-либо другой подсистемы сервера. Замену отказавшего диска и восстановление дискового массива обычно можно выполнить относительно быстро. Так что время работы на зеркальном сервере, вероятно, также будет не очень большим. Кроме того, в пользу зеркального отображения баз данных говорит тот факт, что в отличие от кластеров, переключение обратно на основной сервер происходит очень быстро (обычно несколько секунд). Это достигается за счет того, что зеркальная база данных постоянно поддерживается в готовности. В случае кластеров, процесс восстановления (recovery) базы данных может стать серьезной проблемой.

Еще раз подчеркну, что зеркальное отображение баз данных – это решение для обеспечения высокого уровня доступности на уровне базы данных. Если ваши требования шире, то вы должны либо самостоятельно реализовывать недостающую функциональность, либо задействовать решение на базе кластера. В последнее время в аппаратный лист совместимости(Hardware Compatibility List, HCL) было внесено много самых последних разработок в этой области от разных производителей (например, такого как HP/Compaq). Поэтому внедрение решения на базе отказоустойчивого кластера становится сейчас проще и дешевле, чем когда бы то ни было.

Однако если вы решили пойти по этому пути, то желательно использовать только «полные» кластерные решения. То есть не просто сервер, который сертифицирован для работы в кластере, а сервер, снабженный специальным BIOS'ом, соответствующим SCSI/HBA-адаптером. Должны быть установлены специальные драйвера. Оптический коммутатор (fiber switch) должен быть протестирован и сертифицирован для работы с этой HBA-картой и т.д.

Репликация слиянием изнутри: как идентифицировать ожидающие синхронизации изменения

По материалам статьи Пола Ибизона (Paul Ibison)
«Merge Internals: How to Determine Pending Merge Changes»
Перевод Яна Дмитриевича Либермана

Введение

Я уже давно собирался написать несколько статей посвященных внутренним механизмам репликации слиянием. И сейчас я рад представить вам первую из двух или трех статей для фанатов репликации слиянием, вроде меня.

Одна из задач, решаемых в процессе синхронизации это идентификация изменений, которые должны быть реплицированы. За эту работу отвечает специальный механизм, который выполняется как на издателе, так и на подписчиках. Это достаточно сложный процесс, так как выбираться должны только те изменения, которые еще не были реплицированы. Можно предположить, что реализация этого механизма скрыта где-то в системных хранимых процедурах. Если мы поймем, как этот механизм реализован, то сможем написать запрос, возвращающий список всех изменений, которые будут реплицированы в процессе следующей синхронизации. Для нас, администраторов репликации слиянием — это очень полезная информация, поскольку она позволяет прогнозировать временные характеристики репликации, а также помогает понять характер изменения наших данных.

Для случая репликации транзакций, подобную информацию можно легко получить при помощи системной хранимой процедуры sp_browsereplcmds и виртуальной таблицы MSdistribution_status. Но для репликации слиянием эквивалента этим инструментам до сих пор нет[1].

[1] Начиная с версии SQL Server 2005, стала доступна системная хранимая процедура sp_showpendingchanges. Она возвращает результирующий набор, который отражает примерное количество изменений, ожидающих репликации. Эта хранимая процедура выполняется на издателе в базе данных публикации и на подписчике в базе данных подписки. *Примеч. перев.*

Чтобы заполнить этот пробел, я написал процедуру, которая возвращает интересующую нас информацию (впоследствии мой код дорабатывали Patrick Molijn и Daniel Kristensen). Полный текст процедуры приведен в конце статьи, в листинге 1 (последнюю версию можно найти на сайте www.replicationanswers.com). Эта статья поясняет основные моменты, которые понадобятся, если вы захотите доработать или написать собственную хранимую процедуру, возвращающую информацию об ожидающих репликации изменениях.

Для отслеживания изменений в опубликованных таблицах используются триггеры и системные таблицы. В этой статье мы рассмотрим только те системные таблицы, которые задействованы в механизме отслеживания изменений и оставим без внимания остальные. Кроме того, чтобы сделать статью более понятной, я ограничусь случаем, когда к реплицируемой статье не применен фильтр.

Как узнать, что поменялось?

Для того чтобы на примере посмотреть, как работают внутренние механизмы репликации слиянием, я создал таблицу tCompany, а затем настроил для нее репликацию. Я не хочу чтобы эта статья превратилась в перечень таблиц и процедур, в которых невозможно ничего понять, поэтому далее приводится диаграмма, иллюстрирующая процесс отслеживания изменений в таблице:

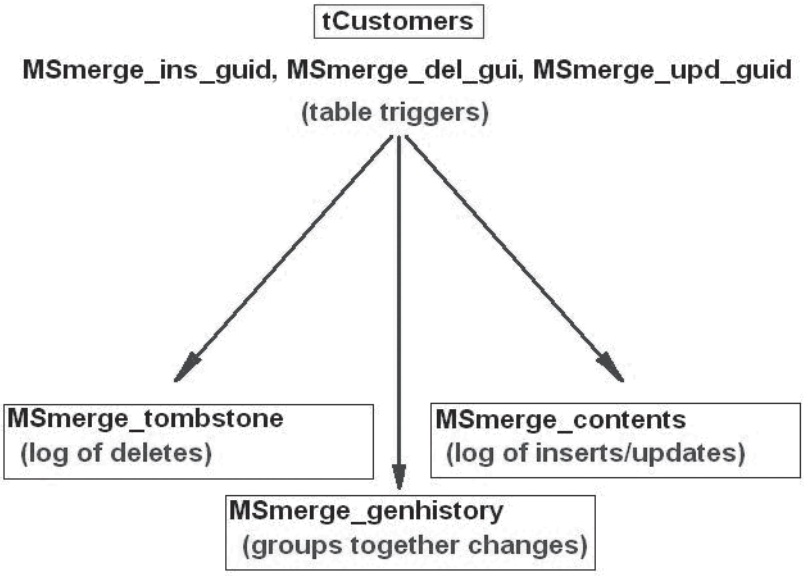


Рис. 1.

После того, как синхронизация завершила свою работу, я исследовал мою таблицу на подписчике. Как и следовало ожидать, в таблицу был добавлен новый столбец с именем rowguid. Он предназначен для хранения уникального идентификатора каждой строки. Также для таблицы было создано три триггера с именами в формате MSmerge_ins/upd/del_6B2966A1FADF41D2B40DF7F8EC131339. Этот странно выглядящий номер в конце имени триггера, на самом деле представляет собой внутренний номер статьи для таблицы tCustomers. Для того, чтобы связать этот номер и название таблицы можно сделать запрос по таблице sysmergearticles (select name, artid from sysmergearticles). Триггеры выполняют простую функцию – если строка добавляется или изменяется, значение колонки rowguid добавляется в таблицу MSmerge_contents. Если строка удаляется, то значение колонки rowguid вставляется в таблицу MSmerge_tombstone (и удаляется из таблицы MSmerge_contents, если она была добавлена туда ранее). Единственный хитрый момент заключается в том, что изменения должны быть сгруппированы. Это делается с помощью так называемых «поколений». Поколение – это коллекция изменений, доставляемых издателю или подписчику. Таблица MSmerge_genhistory содержит одну строку для каждого поколения. Надеюсь, вы уже догадались, как все это работает вместе. Если нет, то давайте с помощью следующей диаграммы, рассмотрим, как выполняются операции вставки и изменения. Это, надеюсь, все прояснит:

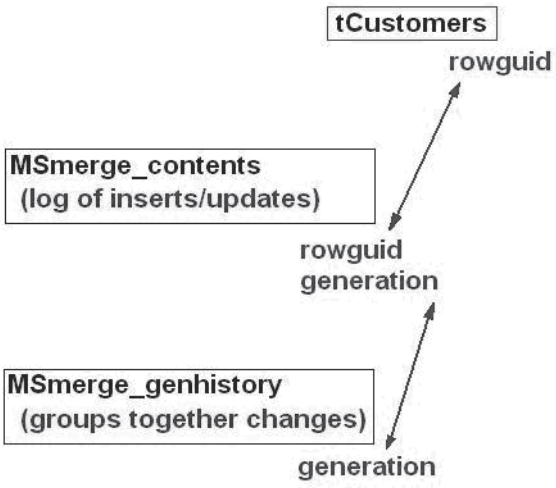


Рис. 2.

Итак, мы знаем, как определить какие строки в таблице tCustomers изменились, и к какому поколению эти изменения относятся. Мы просто должны объединить три упомянутые выше таблицы, например, с помощью следующего запроса:

```
select tCustomers.*
from tCustomers
inner join MSmerge_contents on
    tCustomers.rowguid = MSmerge_contents.rowguid
inner join MSmerge_genhistory on
    MSmerge_contents.generation = MSmerge_genhistory.generation
```

Как выбрать изменения, которые не были реплицированы ранее?

Запрос, приведенный выше, достаточно примитивен. Он всегда возвращает ВСЕ изменения, произведенные в опубликованной таблице. Изначально он будет работать правильно, но если данные в таблице меняются часто, то через некоторое время он вернет всю таблицу целиком, что, как очевидно, бесполезно. Количество строк, возвращаемых этим запросом, будет постоянно расти вплоть до момента, когда будет выполнена очистка метаданных. После этого он вернет пустую таблицу. То, чего я хочу достичь — это найти метод нахождения только тех изменений, которые еще не были реплицированы. Для этого нам нужно обратить внимание на номера поколений.

Первоначально поколения создаются «открытыми» (genstatus = 0). После запуска синхронизации в системной хранимой процедуре sp_MSmakegeneration статус поколения меняется на «закрытое» (genstatus = 1). Это предотвращает любые параллельные изменения этих поколений, в то время как агент репликации слиянием выполняет процедуру sp_MSenumchanges для идентификации изменений, которые должны быть реплицированы. Процедура sp_MSmakegeneration также разбивает большие поколения на несколько поколений меньшего объема (по 1000 изменений в каждом). Это делается для того, чтобы, в случае если работа агента прерывается, можно было продолжить с текущего места, а не выполнять всю работу заново.

Агент репликации слиянием должен различать недавно закрытые поколения и поколения, которые закрыты давно. Делает он это с помощью столбцов sentgen и recgen из таблицы sysmergesubscriptions. К счастью, наш случай несколько проще, потому что мы проверяем данные до того, как начнется синхронизация. Нам достаточно просто отбирать «открытые» поколения, как, например, в следующем запросе:

```
select tCustomers.* from tCustomers
inner join MSmerge_contents on tCustomers.rowguid = MSmerge_contents.rowguid
inner join MSmerge_genhistory on MSmerge_contents.generation =
MSmerge_genhistory.generation
where MSmerge_genhistory.genstatus = 0
```

Чтобы определить количество изменений в каждом поколении, мы можем воспользоваться конструкцией GROUP BY. Однако есть способ

проще. Эту информацию можно извлечь из колонки changecount таблицы MSmerge_genhistory.

```
select changecount from MSmerge_genhistory
where MSmerge_genhistory.genstatus = 0
```

Собираем все вместе

Листинг 1 содержит код процедуры spBrowseMergeChanges. Первоначально эта процедура была написана мной, а затем Patrick Molijn и Daniel Kristensen в течение года или около того дорабатывали ее. Процедура возвращает список изменений, которые ожидают репликации. Так как запрос в процедуре адаптирован для реальных условий, он выглядит сложнее, чем тот, который приводился ранее. Однако он построен абсолютно на тех же принципах. Код в начале процедуры (первые пять строк) нужен для определения имени столбца с установленным свойством «IsRowGUIDCol» (этот столбец не всегда называется «rowguid»). Динамический запрос в процедуре используется, в связи с тем, что имя статьи не «вшито» в процедуру, а передается как параметр. И, наконец, конструкция union all применена, потому что мы хотим различать операции вставки/изменения и операции удаления. Все это дополнительные улучшения, но как я уже говорил ранее, основная идея запроса осталась неизменной.

Листинг 1.

```
create Proc spBrowseMergeChanges (@articlename sysname)
/*****************************************************
Создана: Paul Ibison (www.replicationanswers.com)
Изменена: Patrick Molijn (patrick@molijn.net). Поддержка столбцов rowguid, имеющих
название, отличное от ROWGUID - 23/5/2005
Изменена: Patrick Molijn (patrick@molijn.net). Для SQL 2005 - 06/5/2006
Изменена: Patrick Molijn (patrick@molijn.net). Использование столбца genstatus SQL
2005 - 27/5/2006
Дата: 27 Мая 2006
Применение: Перечисление всех изменений на подписчике, которые еще не были
реплицированы.
*****************************************************/
as
begin
-- Получить название столбца rowguidcol статьи
declare @ROWGUIDCol sysname
select @ROWGUIDCol = c.[name]
from syscolumns c
where c.id = object_id(@articlename)
and columnproperty(object_id(@articlename), c.[name], 'IsRowGUIDCol') = 1

if @ROWGUIDCol is null
return – Не удается определить rowguidcol, так что вернем null
```

```
exec ('SELECT ''Ins/Upd'' as Type,
mc.rowguid AS RowGuid,
smp.publisher AS Publisher,
smp.name AS Publication,
sma.name AS Article,' + @articlename + '.*
from dbo.MSMerge_genhistory mh with (nolock)
inner join dbo.MSmerge_contents mc with (nolock) on (mc.generation = mh.generation)
inner join ' + @articlename + ' with (nolock) ON mc.rowguid = ' + @articlename + '. ' +
@ROWGUIDCol
+ ' inner join dbo.sysmergearticles sma with (nolock) on (sma.nickname = mc.tablenick)
-- Наши изменения
inner join dbo.sysmergesubscriptions sms with (nolock) on (sms.pubid = sma.pubid and
sms.subid != sms.pubid )
-- Кому необходимо их передать
inner join dbo.sysmergesubscriptions smReceiver with (nolock) on (smReceiver.subid =
sms.pubid)
inner join dbo.sysmergepublications smp with (nolock) on (smp.pubid =
smReceiver.pubid)
inner join dbo.MSmerge_replinfo mri with (nolock) on (sms.subid = mri.repid )
where mh.genstatus = 0
and sma.name = ''' + @articlename + '''

union all
SELECT ''Del'',
mt.rowguid,
smp.publisher,
smp.name,
sma.name, ' + @articlename + '.*
from dbo.MSMerge_genhistory mh with (nolock)
inner join dbo.MSmerge_tombstone mt with (nolock) on (mt.generation = mh.generation)
inner join dbo.sysmergearticles sma with (nolock) on (sma.nickname = mt.tablenick)
-- Наши изменения
inner join dbo.sysmergesubscriptions sms with (nolock) on (sms.pubid = sma.pubid and
sms.subid != sms.pubid)
-- Кому необходимо их передать
inner join dbo.sysmergesubscriptions smReceiver with (nolock) on (smReceiver.subid =
sms.pubid)
inner join dbo.sysmergepublications smp with (nolock) on (smp.pubid =
smReceiver.pubid)
inner join dbo.MSmerge_replinfo mri with (nolock) on (sms.subid = mri.repid )
left outer join ' + @articlename + ' on (mt.rowguid = ' + @articlename + '.' +
@ROWGUIDCol + ')'
+ ' where mh.genstatus = 0
and sma.name = ''' + @articlename + ''''')
end
go
```

Заключение

Надеюсь, вы согласитесь с тем, что идентификация ожидающих репликации изменений для случая репликации слиянием, не такая сложная задача, как, возможно, казалось вначале. Теперь вы сможете создавать свои собственные запросы для мониторинга и устранения неполадок в репликации слиянием.

Усовершенствования очистки метаданных в SQL Server 2005

По материалам статьи Найджел Мейнффа (Nigel Maneffa) «Metadata Cleanup Improvements in SQL2005»

Перевод Александра Сергеевича Волока

Еще больше таблиц метаданных в SQL Server 2005

Количество таблиц метаданных для репликации слиянием постоянно растет. Наряду с таблицами *msmerge_contents*, *msmerge_tombstone* и *msmerge_genhistory*, существующими еще со времён SQL Server 7.0 и SQL Server 2000, появились еще три дополнительные, потенциально важные таблицы. Несомненно, это наводит мысли на то, что количество мест для очистки метаданных в новой версии выросло.

Базовые таблицы с метаданными, необходимые для репликации слиянием:

MSmerge_contents
MSmerge_tombstone
MSmerge_genhistory

Вспомогательные таблицы с метаданными, необходимые для предварительно вычисляемых секций:

MSmerge_current_partition_mappings
MSmerge_past_partition_mappings (Transact-SQL)
MSmerge_generation_partition_mappings (Transact-SQL)

Таблица Msmerge_generation_partiotion не документирована в электронной документации по SQL Server (BOL), но очищается при вызове хранимой процедуры sp_mergemetadataretentioncleanup.

Меня заставляет вздрогнуть мысль о том, что теперь необходимо очищать еще больше метаданных – потенциально увеличивая время синхронизации относительно SQL Server 2000. Все же, я решил найти больше информации о характере использования этих таблиц, и действительно ли время очистки метаданных увеличится.

Как бы то ни было, чтобы получить информацию о размерах этих таблиц на своём промышленном сервере, я использовал системную хранимую процедуру sp_spaceused, отфильтровав ненужные данные:

	Количество строк	Размер
MSmerge_contents	465245	481672 КБ
MSmerge_tombstone	44687	11072 KB
MSmerge_genhistory	99601	28000 KB

Новые таблицы для предварительно рассчитанных сегментов:

	Количество строк	Размер
MSmerge_current_partition_mappings	734865(!!)	148800 KB
MSmerge_past_partition_mappings	9045	3744 KB
MSmerge_generation_partition_mappings	206105	13760 KB

Нововведения очистки метаданных в SQL Server 2005

Когда я начал тестирование SQL Server 2005, я был удивлен тем, насколько рационально и регулярно выполняется здесь очистка метаданных, по сравнению с SQL Server 2000. Процедура очистки метаданных в SQL Server 2000 могла стать причиной резкого увеличения продолжительности синхронизации, причём, практическая ценность самой очистки не велика. Сам же процесс очистки был очень требователен к дисковым операциям ввода-вывода и интенсивно использовал процессорные ресурсы, даже в случае небольшой по объёму очистке метаданных. В любом случае, меня интересовало, корректно ли я отслеживал изменения? Или же SQL Server 2005 в тестовом варианте среды не позволил мне получить реальные результаты? Далее, разбирая полученные в процессе исследований результаты, я обнаружил, что в системной таблице sysmergesubscriptions появился новый столбец, содержащий дату и время последней очистки метаданных – возможность, которая выглядит многообещающей.

```
SELECT metadatacleanuptime,* FROM  sysmergesubscriptions WITH (NOLOCK)
```

Запуск приложения SQL Server Profiler оправдал эти надежды – вот фрагмент одной из команд, выполняемых при синхронизации:

```
.....@dometadata_cleanup = case when sys.fn_add_units_to_date(-1,
@retention_period_unit, getdate()) > metadatacleanuptime then 1....
```

В Microsoft решили навести порядок с проблемами очистки метаданных, для этого они ввели понятие единицы срока хранения ('retention units'), которую задаёт администратор (читайте об этом далее). Это очень простое и изящное решение. В предыдущих версиях очистка метадан-

ных запускалась в самом начале каждого запланированного сеанса синхронизации репликации слиянием. Приятной новостью стало то, что в новой версии предусмотрено управление временем выполнения очистки метаданных, что делает этот процесс предсказуемым и менее затратным с точки зрения ресурсов сервера. По умолчанию срок хранения установлен на 14 дней, очистка метаданных планируется раз в день, и может выполняться в каждом сеансе синхронизации, если синхронизация выполнялась раз в день, что является приемлемым для большинства пользователей в большинстве случаев.

Похоже на то, что разработчикам репликации слиянием не оставалось другого выбора, поскольку для оптимального выполнения очистки метаданных, включая новые таблицы (имеющие потенциально большое количество записей для операторов просмотра плана исполнения запроса), старый механизм мог оказаться слишком затратным. Более того, если заглянуть в детали процесса синхронизации с помощью приложения SQL Server Profiler, можно заметить, что выполнялись и другие (недокументированные) процессы очистки. Хорошая новость для нас в том, что они выполняются так же оптимально.

Периоды единиц сохранности в SQL Server 2005

Кстати, сейчас стало возможным иметь разные единицы измерения для сроков хранения. По умолчанию, это, как и было раньше – дни (в SQL Server 2000 была только одна единица измерения). Но также возможно указывать недели, месяцы и даже годы, если уровень совместимости базы данных публикаций не ниже 90RTM (согласно BOL). Если для публикации используется база данных с уровнем совместимости ниже 90RTM, тогда «дни» будут единственной опцией в качестве единиц срока хранения метаданных.

В необновлённой версии BOL по этому поводу есть неточность. Там говорится, что значения retention_period_unit равны: 0=день, 1=неделя, 2=месяц и 3=год. Но также существуют недокументированные значения 4 и 5, которые означают часы и минуты, и установка их возможна посредством пользовательского интерфейса.

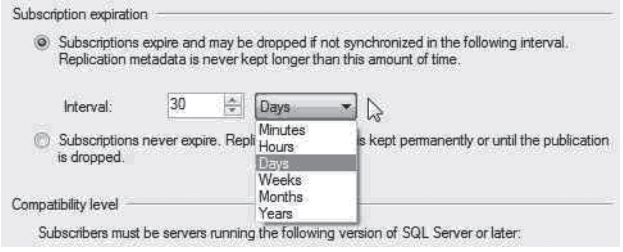


Рис. 1.

Ручной запуск очистки метаданных, не используя sp_mergemetadataretentioncleanup

При необходимости выполнять очистку метаданных чаще, чем это можно делать по расписанию (я не могу представить, зачем это может понадобиться) можно вручную изменять значение в поле sysmergesubscriptions.metadatacleanuptime, что позволит обмануть подсистему репликации, заставляя ее полагать, что очистка метаданных проводилась давно.

На моей тестовой системе (все подписчики имеют одинаковый срок хранения, равный 14 дням) ручная очистка отлично работает. Я лишь перенес дату последней очистки на один день дальше срока хранения, и подсистема репликации очистила метаданные во время следующего сеанса синхронизации, установив время очистки на текущий момент. Однако я рекомендую с осторожностью использовать подобные методы в промышленной среде, и каждый раз проводить предварительное тестирование, если понадобилось выполнять очистку метаданных чаще, чем это можно сделать по расписанию. Здесь же уместны и стандартные для подобных случаев замечания, например о том, что внесение изменений в системные таблицы не рекомендуется Microsoft. Этого не стоит делать потому, что не исключена возможность внесения некорректных данных, последствия чего могут быть непредсказуемы, вследствие чего подобные методы могут применяться только на свой страх и риск. Рекомендуемой практикой является ручной запуск системной процедуры sp_mergemetadataretentioncleanup. Однако, техника «установки устаревшей даты», по моему мнению, является более простой, поскольку рекомендуемую хранимую процедуру необходимо выполнять дважды, как на издателе та и на подписчике. Заметьте, если у вас используются подписчики более ранней версии, нежели SQL Server 2000 SP1, нужно использовать другую хранимую процедуру sp_mergecleanupmetadata.

Итоги

В промышленной среде, о которой я упоминал в этой статье, расписание сеансов синхронизации было выбрано раз в 15 минут, публикации не имели фильтров, и число синхронизуемых в одном сеансе записей было небольшим. Время синхронизации снизилось с 15 секунд в SQL Server 2000 до 2 секунд в SQL Server 2005. Нагрузка на систему тоже значительно уменьшилась. Главной причиной этого стала оптимизация в новой версии расписания очистки метаданных.

Выводы

Команда разработчиков репликации сделала огромную работу по оптимизации в SQL Server 2005 алгоритмов очистки метаданных. Прежние пути решения связанных с этим проблем, которыми иногда приходилось пользоваться для увеличения производительности системы, уже неактуальны. В моем случае, новая методика экономит как минимум 100 000 000 логических чтений в день (я не ошибся количеством нулей!), которые были пустой тратой ресурсов.

Репликация SQL Server 2005/2008

Сборник статей от сообщества SQL.RU

Под общей редакцией
Александра Гладченко и Владислава Щербинина

Верстка *М. Алексеевой*
Дизайн и оформление *О. Будко*

Директор издательства *В. Говорухин*
Главный редактор *Л. Захарова*

Подписано в печать 18.22.2008. Формат 70x100 $^{1}/_{16.}$
Гарнитура Журнальная. Печать офсетная. Усл. печ. л. 23,33
Тираж 2000 экз. Заказ №

ЭКОМ Паблишерз
117342, Россия, Москва, ул. Бутлерова, д.17а, оф. 105
Телефон для оптовых покупателей (495) 330-68-65