Организация баз данных
2005. №1(9)

УДК 004.65

© 2005 г. **А.Д. Плутенко,** канд. техн. наук,
А.А. Ситников
(Амурский государственный университет, Благовещенск)

ПРОБЛЕМЫ ИМИТАЦИОННОГО МОДЕЛИРОВАНИЯ ПРОЦЕССА ЖУРНАЛЬНОЙ РЕПЛИКАЦИИ ДАННЫХ В РАСПРЕДЕЛЕННЫХ СУБД[1]

В работе рассматриваются характеристика метода журнальной синхронизации, задачи, возникающие при использовании репликации по журналу, а также проблема идентификации записей для нескольких участников репликационного обмена.

Введение

В последнее время процесс развития телекоммуникационных сетей, систем передачи данных, развитие инструментальных средств разработки и увеличившийся спрос заказчиков информационных систем на такие качества как доступность и надежность создаваемых систем позволяют разработчикам программных систем создавать распределенные гетерогенные приложения, отвечающие потребностям бизнеса в доступности данных. Не последнюю роль в технологиях создания подобных систем играет реализация процесса синхронизации состояний узлов в распределенной базе данных.

Распределенная база данных – это база данных, которая не сосредоточена в одном месте, а расположена на некотором множестве узлов (или серверов баз данных). Обычно при реализации подобной схемы каждый из пользователей соединен со своим, местным сервером СУБД, а данные передаются между БД с применением какого-либо механизма синхронизации, обычно называемого репликацией (реплика – отдельный атомарный элемент передаваемых данных). То есть система распределенных баз данных состоит из набора узлов, связанных коммуникационной сетью. В ней

[1] Работа выполнена в рамках плана НИР по заданию Федерального агентства по образованию в 2005 г. «Исследование характеристик производительности распределенных систем обработки данных на ранних стадиях проектирования».

каждый узел – это полноценная СУБД, но узлы взаимодействуют между собой таким образом, что пользователь любого из них может получить доступ к любым данным в сети так, как будто они находятся на его собственном узле.

Отметим, что каждый узел сам по себе является системой базы данных. Иначе говоря, на каждом узле есть собственные локальные "реальные" базы данных, собственные локальные пользователи, собственные локальные СУБД и программное обеспечение управления транзакциями, собственный локальный менеджер передачи данных. В частности, любой пользователь может выполнить операции над данными на своем локальном узле точно так же, как если бы этот узел вовсе не входил в распределенную систему. Распределенную систему баз данных можно рассматривать как некоторое партнерство между отдельными локальными СУБД на отдельных локальных узлах. Пример подобной структуры показан на рис. 1.

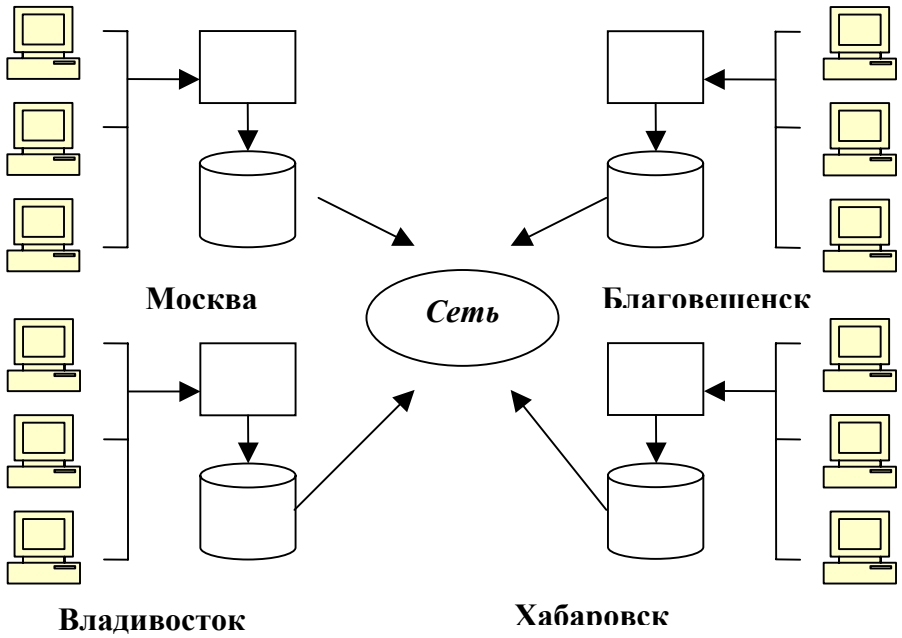

Рис 1.

Из этого определения следует, что так называемая "распределенная база данных" в действительности является виртуальной базой данных, компоненты которой физически хранятся в нескольких "реальных" базах данных на нескольких различных узлах (в сущности, являясь логическим объединением этих реальных баз данных).

Зачем нужны распределенные базы данных? Основная причина заключается в том, что предприятия обычно уже распределены. Распределены по крайней мере логически, т.е. разделены на подразделения, отделы, рабочие группы и т.д. Очень часто они распределены и физически. Из этого следует, что данные также обычно распределены, поскольку каждая организационная единица создает и обрабатывает собственные данные, относящиеся к деятельности этой единицы. Таким образом, информация пред-

приятия разбивается на части, которые иногда называют островами информации. А распределенная система обеспечивает мосты для их соединения в единое целое.

Репликация желательна по крайней мере по двум причинам. Во-первых, она способна обеспечить более высокую производительность, поскольку приложения смогут обрабатывать локальные копии вместо того, чтобы устанавливать связь с удаленными узлами. Во-вторых, наличие репликации может обеспечивать также более высокую степень доступности, поскольку любой реплицируемый объект остается доступным для обработки (по крайней мере для выборки данных), пока хотя бы одна реплика в системе остается доступной.

Репликацией называется процесс приведения неодинаковых состояний двух и более баз данных со схожей структурой в одинаковое состояние, удовлетворяющее заданным критериям процесса репликации с сохранением условия правильности БД. Состояние базы данных при этом определяется набором ее данных и структур. Применимо к реляционным СУБД можно сказать, что состояние БД описывается структурой таблиц и содержащимися в них данными в определенный период времени.

Краткая характеристика метода журнальной синхронизации

Репликация по журналу – это наиболее простой в реализации метод и один из наиболее эффективных. При реализации процесса синхронизации двух и более БД на каждой из них повторяют все действия, произведенные в другой БД, извлекая их по очереди из журнала. Отсюда вытекает одно из основных преимуществ синхронизации по журналу – нет необходимости в постоянном соединении с удаленной базой данных во время передачи изменений – часть журнала является автономной единицей, ее можно передать по любым каналам связи в любое время, в том числе даже на дискете. То есть журнал обладает свойством переносимости, которое достаточно трудно реализуется при использовании других методов.

Для реализации этого метода необходимо тщательно отмечать, какие записи журнала еще не передавались. Эта проблема усложняется при наличии нескольких синхронизируемых БД – нужно запоминать, какая последняя запись журнала была передана в каждую из БД.

Недостатком использования журнала является то, что журнал занимает много места. При достаточно частом изменении таблиц баз данных и больших промежутках времени между сеансами репликации журнал может достигать больших объемов и даже превосходить по размеру сами данные, хранящиеся в БД. Понятно, что после успешного применения журнала к другой БД журнал можно урезать. Но, тем не менее, журнал занимает много места – в лучшем случае дублирует новые и измененные записи.

Журнальная синхронизация реализуется достаточно просто, но коллизии (конфликты) при этом неизбежны, так как процесс, проводящий ре-

пликацию, обычно не имеет одновременного доступа к двум базам данных.

Проблемы, возникающие при использовании репликации по журналу

Принцип журнальной репликации достаточно прост – в базе данных ведется журнал, который через определенные промежутки времени передается другой базе для повторного воспроизведения всех выполненных в первой базе данных действий. Следовательно, для каждой из локальных баз данных (для каждого из узлов) характерно наличие хотя бы одной из двух следующих ролей: роль источника репликационной информации (источник) и роль приемника репликационной информации (приемник).

Источник – узел распределенной БД, создающий поток репликационной информации (журнал) для отражения ее на других узлах. В журнальной репликации узел-источник ведет журнал, в который записываются действия пользователей, а затем этот журнал передается узлам-приемникам для воспроизведения.

Приемник – узел распределенной БД, отражающий все действия, произведенные в узлах-источниках, принимающий поток репликационной информации.

В терминологии репликации MS SQL Server, источник – это издатель (publisher), а приемник – подписчик (subscriber). Возможно (и наиболее распространено) совмещение двух этих ролей на одном узле. При этом возможно построение любых, достаточно сложных схем распределенной базы данных, нужно лишь правильно определить пути распространения репликационной информации и роль узлов (источник, приемник). Рассмотрим некоторые типичные проблемы, которые могут возникать при взаимодействии узлов распределенной базы данных при использовании журнальной репликации.

Общие проблемы журнальной репликации независимо от конфигурации распределенной БД

Рассмотрим проблемы, которые имеют общий характер независимо от конфигурации распределенной БД и состава узлов участников. Можно выделить следующие общие проблемы репликации.

1. Все локальные БД узлов-участников распределенной БД при репликации должны иметь одинаковую структуру реплицируемой части базы данных. Они должны иметь один и тот же состав таблиц, триггеров, хранимых процедур в той области, которую необходимо реплицировать. Однако БД могут иметь разные структуры в части нереплицируемых данных (например, можно содержать какие то данные, специфические для каждого из узлов, если их не нужно синхронизировать между узлами). Тогда решение задачи репликации сводится к отражению изменений данных узла-источника на узле-приемнике в части БД или в целом. Если же БД будут

иметь разные структуры, нужно будет вводить правила отражения одной структуры на другую. Для поддержания структур узлов в одинаковом состоянии необходимо разрабатывать какие-либо технические средства или административные правила распространения изменений структур баз данных.

2. Любая база данных всегда содержит бизнес-логику, предназначенную для соблюдения определенных правил, форматов данных и т.п. Бизнес-логика выполняется с помощью таких средств как триггеры и хранимые процедуры. Предположим, есть какой-то триггер, который при вставке записи в таблицу генерирует значение одного из полей (например, присваивает одному из полей значение текущего системного времени). После выполнения вставки в журнал будет помещена запись об этом изменении. При выполнении журнала на узле-приемнике и, соответственно, при вставке этой записи уже на этом узле триггер выполнится еще раз (и для записи будет сгенерировано другое значение генерируемого поля). Для примера с полем, в которое записывается системное время источника, для приемника будет записано уже другое значение системного времени, и данные будут искажены. Иллюстрация данной ситуации приведена на рис. 2.

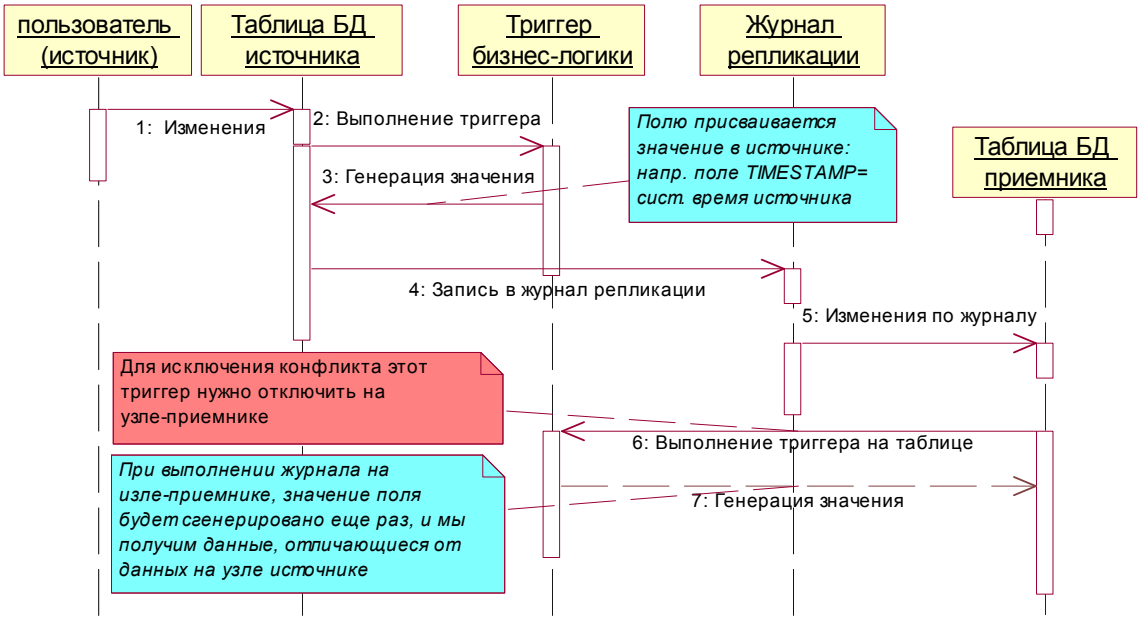

Рис. 2.

Такая же проблема возникнет, если триггер изменяет данные в других таблицах базы данных. Так как эти изменения все равно собираются в журнал репликации, то при включенном триггере БД приемника одни и те же действия будут выполнены повторно, что может привести также к искажению данных.

Для предотвращения описанной ситуации необходимо произвести анализ бизнес-логики с целью поиска и устранения похожих ситуаций. Можно отключать триггеры, генерирующие значения на БД узла-приемника, но это не всегда возможно, так как с базой могут работать дру-

гие пользователи, которым данные триггеры необходимы. Поэтому обычно применяется модификация триггеров с тем, чтобы они анализировали, в результате чего производится изменение данных – в результате действий локальных пользователей или в результате выполнения журнала от БД-источника. В случае выполнения триггера от действий пользователя он выполняется полностью, в противном случае – пропускает места, связанные с присваиванием значений изменяемой записи либо с изменением данных в других таблицах. Можно заметить, что триггеры, занимающиеся проверкой данных и любыми действиями, не связанными с модификацией данных, отключать необязательно – не имеет значения, если какая-либо проверка выполнится и на узле-источнике и на узле-приемнике. В любом случае необходим тщательный анализ всей бизнес-логики для подготовки БД к репликации.

Специфические проблемы журнальной репликации, связанные с конфигурации распределенной БД

Теперь рассмотрим проблемы репликации по журналу, связанные с конфигурацией распределенной базы данных, с расположением узлов и с назначенной каждому узлу ролью.

Рассмотрим сначала самый простой случай – однонаправленной репликации. В этом случае есть один источник и один приемник журнала, то есть фактически осуществляется просто копирование данных с одной БД в другую и выполняется поддержание копии БД. Схема такой односторонней репликации представлена на рис. 3.

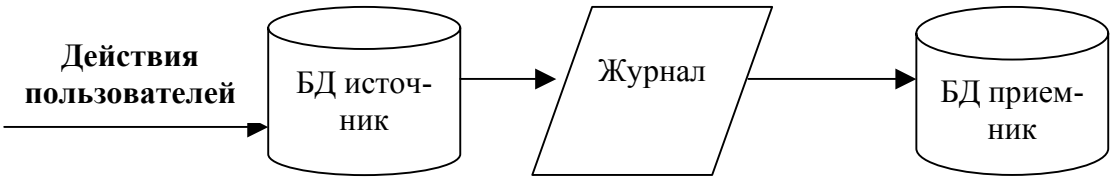

Рис. 3.

В этом случае каких-то специфических проблем обычно не возникает. При разрешенных общих проблемах по одинаковой структуре и бизнес-логике процесс выглядит достаточно просто и тривиально: действия пишутся в журнал, журнал потом выполняется на удаленном узле-приемнике. Кстати, при этой схеме узлов-приемников может быть сколь угодно много, неважно, сколько раз выполнится один и тот же журнал. База данных, которая получатся в узлах-приемниках, представляет собой просто теневую копию БД в узле–источнике.

Теперь изучим проблемы, которые могут возникать, если реализовывать схему с двусторонней репликацией, то есть когда узлы являются и источником и приемником репликационной информации. Рассмотрим случай

только для двух участников. Схема подобной репликации дана на рис. 4.

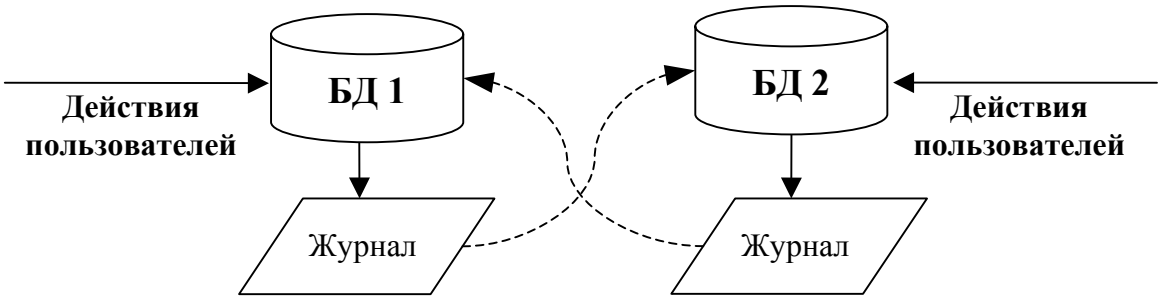

Рис. 4.

При подобной схеме реализации репликации данных каждый из узлов-участников процесса выступает в качестве и источника, и приемника репликационных данных. В приведенной схеме могут возникать следующие проблемы.

Во-первых, каждый из узлов получает журнал действий, выполненных на другой стороне. Каждый узел принимает запросы на модификацию данных как от локальных пользователей, которые работают с базой данных, так и от удаленной стороны репликации. При воспроизведении этого журнала узлом-приемником выполняются все записанные действия, которые были выполнены на узле- источнике. Если этот узел-приемник выступает в роли также узла-источника, то данные, выполненные в процессе воспроизведения журнала, также попадут в журнал узла-приемника и, соответственно, при следующем сеансе связи передадутся обратно отправителю. То есть может возникнуть процесс неконтролируемого лавинообразного процесса накопления репликационной информации, когда узлы передают друг другу информацию об одной и той же записи

Для предотвращения подобного поведения можно сделать следующее: при приеме и выполнении журнала, полученного от другой стороны, – не вести собственный журнал, чтобы изменения, прошедшие на другой стороне, были выполнены только над данными и не попали в журнал, предназначенный для отправки обратно узлу, приславшему данную репликацию. Если для реализации журнальной репликации используется механизм триггеров, то нужно либо блокировать их выполнение во время выполнения журнала другой стороны, либо написать их таким образом, чтобы они распознавали момент выполнения журнала другой стороны и не писали бы в таком случае в свой журнал.

Если нет блокировки ведения журнала информации, полученной от партнера репликации, возникает ситуация, представленная на рис. 5, т.е. репликационная информация ходит от одного узла к другому, что, конечно же, недопустимо.

Во-вторых, при реализации двусторонней репликации неизбежно возникает проблема конфликта доступа к одним и тем же данным на двух

сторонах участников репликационного процесса.

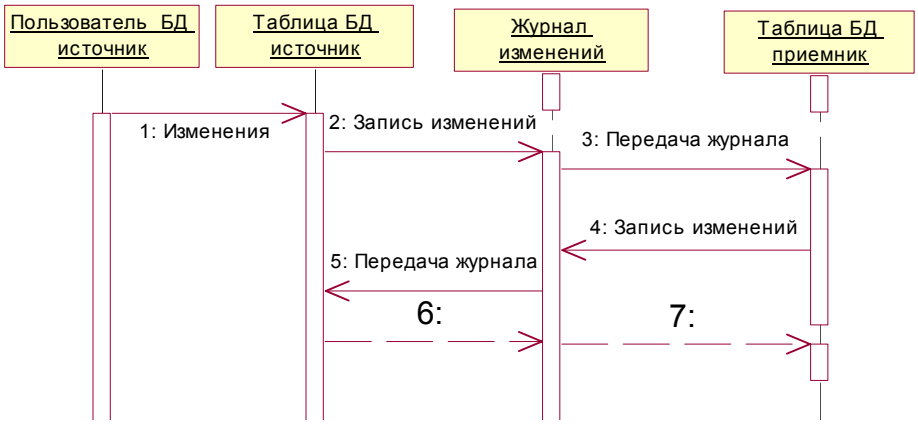

Рис. 5.

Этот конфликт возникает в том случае, если на двух узлах осуществляются изменения одних и тех же данных. Допустим, на узле 1 создали запись, затем на узле 2 ее удалили, а в то же время была выполнена операция изменения этой записи на узле 1. Естественно, возникает конфликт, который требует решения – какие изменения считать правильными. Основной вопрос, на который нужно дать ответ при возникновении подобного рода конфликтов, это вопрос – какие данные все же верные, кто из узлов-участников прав. Заметим, что не всегда возможно принять автоматизированное решение о предпочтении изменения для какой то записи, и требуется решение человека.

Обычно для разрешения этого конфликта используется несколько основных предпосылок:

более верными считаются данные у того из узлов, который изначально создал запись, кто «считается ее владельцем». Для реализации этого правила необходимо, чтобы в каждой из таблиц репликационного процесса содержалось поле, указывающее на узел-автора данной записи;

вводится дополнительное поле к журналу – «время выполнения изменения». При выполнении журнала время изменения записи сравнивается со временем локального изменения этой записи, более «правильной» представляется информация, которая получена позже. Отметим, что это не самый надежный способ разрешения конфликтов, так как может образовать конфликт изменение одной и той же записи, произошедшее в разное время;

один из самых надежных способов – это механизм слежения за возникновением конфликтных ситуаций, в случае возникновения таких ситуаций – требовать решения человека, не решая, какая из сторон права, но предоставляя оценку каждой из таких ситуаций.

При двусторонней репликации в дополнение к общим проблемам, которые не зависят от структуры распределенной БД и роли каждого из узлов (см. выше), добавляются специфичные проблемы, которые харак-

терны только для реализации данного вида репликации. Их можно разделить на две группы: проблемы воспроизведения журнала репликации на удаленном узле, проблемы разрешения конфликтов, возникающих в случае одновременного доступа к одним и тем же данным.

Теперь рассмотрим те проблемы, которые могут возникать при реализации репликации на количество узлов более чем два. Каждый из узлов, участвующих в такой сети, может выполнять роль как источника так и приемника репликационной информации, а может и совмещать эти роли (наиболее часто используемый вариант поведения). В качестве примера приведем следующую схему (рис. 6), на которой изображены 4 базы данных с журналами и средой распространения журнальной информации.

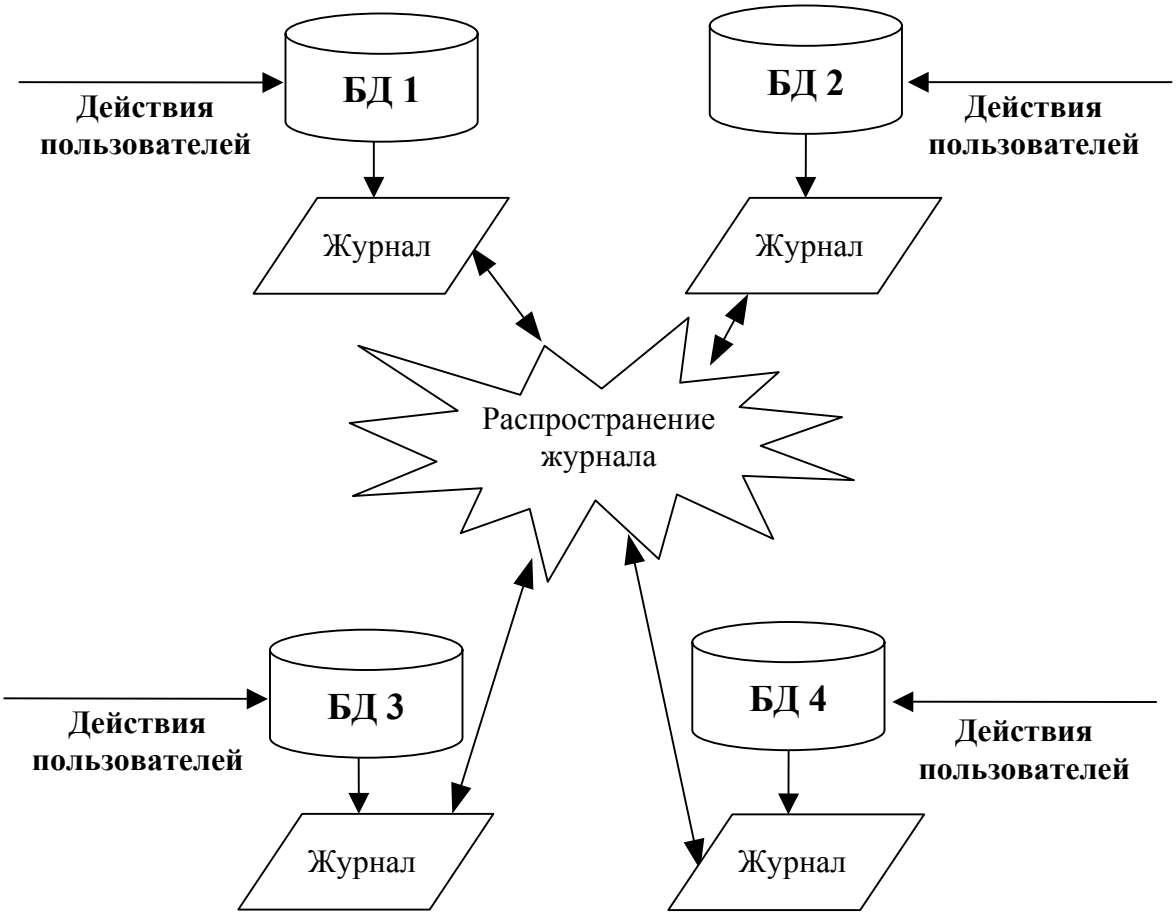

Рис. 6.

В качестве среды распространения журнала репликации подразумевается любая сетевая среда, обеспечивающая доставку журнала каждого из участников всем остальным участникам процесса репликации. Все сделанные пользователями изменения в каждой из БД участников информационного обмена распространяются на каждый из узлов. Рассмотрим, какие проблемы возникают при этом.

Если взять первую проблему, возникшую при реализации процесса двусторонней репликации, то есть когда недопустимо повторное выполнение изменений данных, инициируемых журналом одного узла на узле, породившем данную репликацию. Для случая нескольких узлов задача усложняется – здесь уже не поможет простое блокирование изменений, которые были произведены путем воспроизведения полученного журнала репликации. Необходима идентификация записей журнала – какая из них получена от каждого из участников. При данной схеме нужно устанавливать фильтр на выход журнала от каждой из базы данных и подготавливать репликационный журнал отдельно для каждого участника репликационного обмена по такому критерию: подготавливать все записи журнала, созданные как локальными пользователями, так и созданные в результате выполнения журналов от других участников распределенной базы данных. Но при подготовке данных журнала для какого-то узла обмена необходимо исключать данные журнала, полученные от этого самого участника. Так, если рассматривать схему, изображенную на рис. 6: если БД 2 подготавливает журнал для БД 1, то туда войдут изменения как и БД 2 (локальные пользователи), БД 3, БД 4, так и будут исключены записи журнала, которые были получены от БД 1. Для реализации подобного поведения необходимо в журнале проставлять идентификатор создавшей его базы, при выполнении этого журнала на БД-приемнике записывать оригинальный идентификатор БД создателя в журнал, а при подготовке ответного журнала этой БД исключать полученные от нее записи.

Кроме описанных проблем реализации репликации, существует еще проблема обеспечения логической целостности распределенной БД. Не последнюю роль в этом играет решение проблемы однозначной идентификации записей, созданных в разных узлах одной распределенной базы данных. Рассмотрим эту проблему более подробно.

Проблема идентификации записей для нескольких участников репликационного обмена

В реальной жизни основа любой учетно-финансовой автоматизированной системы – это объекты, которые хранятся в таблицах-справочниках БД. А для любого справочника, как минимум, главное – это поддержание уникальности своих идентификаторов в пределах распределенной базы данных. Обычно идентификатор записи создается автоматически средствами СУБД (для СУБД вида Interbase/Firebird, на использование которых и ориентирована данная статья; значения идентификаторов обычно присваиваются с использованием специальных объектов – генераторов, при помощи триггера Before Insert). Если эту схему перенести на распределенную базу, в которой все базы данных работают независимо друг от друга, и, соответственно, генерируют идентификаторы, то без конфликтов в создании идентификаторов не обойтись.

Нужно отметить, что данная проблема может возникать только в том случае, если идентификаторы записей представляют собой искусственные ключи, не связанные логически с теми данными, которые они идентифицируют. В случае естественных ключей все намного проще, естественные ключи связаны со своими данными и обычно не генерируются автоматически (в качестве примера можно привести идентификацию записей в финансовой системе, в которой запись клиента идентифицируется по полю ИНН). Для подобной системы, естественно, не возникает проблемы генерации ключей, так как ключ сгенерирован вне системы.

Для решения проблемы генерации ключей есть несколько вариантов. Различают два основных варианта: либо использовать какой-то центральный сервер выдачи идентификаторов, которым будут пользоваться все участники информационного обмена для получения уникальных идентификаторов, либо обеспечить каким-то математическим, логическим и иным образом генерацию уникальных идентификаторов в пределах распределенной базы данных без связи баз данных друг с другом.

Для искусственных ключей в основном используется тип данных INTEGER (целочисленный). Числовой идентификатор – наиболее простой и очень распространенный в уже существующих системах, но в качестве ключа можно использовать не только целочисленные номера, можно вводить дополнительные поля или применять другие типы данных. Рассмотрим несколько вариантов решения проблемы идентификаторов записей.

1. Можно ввести в каждую таблицу дополнительное поле – идентификатор БД, в которой эта запись была создана впервые (например, целочисленное поле «DBID»). При этом, очевидно, поле основного ключа (например, «ID») уже не будет первичным ключом, а первичным ключом записей станет пара полей (DBID, ID). Это одно из самых простых решений, но оно плохо применимо в случае больших объемов данных или достаточно большой структуры базы данных. В первом случае будет тратиться много памяти на организацию индекса первичного ключа для большого количества данных. Во втором случае разветвленная структура базы данных может быть препятствием для введения данного решения из-за сложности реорганизации структуры (в каждую таблицу нужно добавить поле). Поэтому следует заметить, что данное решение привлекательно как наиболее простое только в случае небольших объемов данных. Для большого количества данных это решение не слишком привлекательно.

2. Можно сделать первичным ключом строку специального формата, – например, XXXX-YYYY-ZZZZZZZZ, где XXXX – идентификатор базы данных, где запись была создана впервые; YYYY – идентификатор таблицы; ZZZZZZZZ – идентификатор записи внутри конкретной таблицы конкретной БД. Такое решение является хорошо масштабируемым, позволяет вводить в идентификатор много дополнительной информации, но у него тоже есть свои минусы. Во-первых, при данном решении наблюдается

большая избыточность информации (не только из-за повторения некоторых частей строкового идентификатора, но и из-за того, что строка сама по себе занимает больше памяти, чем число). Во-вторых, это достаточно сложный механизм генерации идентификаторов, который нужно обеспечить написанием сложных триггеров, проверок, и.т.п.

3. Можно использовать одно поле типа INTEGER (целочисленное), но обеспечить генерацию идентификаторов таким образом, чтобы идентификаторы, сгенерированные в различных местах распределенной БД, не пересекались. Этого удается достигнуть, если разбить область значения целочисленного идентификатора на несколько непересекаемых областей и административно назначить каждую из областей для использования в каждом из улов распределенной БД. Например, можно назначить административным путем, что для головного офиса диапазон значений идентификатора может составлять 1..1000000, для доп. офиса 1 – 1000001..2000000, для доп. офиса 2 – 2000001..3000001., и, генерируя значения в пределах заданных диапазонов идентификаторы никогда не пересекутся.

Очевидно, что для СУБД InterBase/Firebird одним из лучших вариантов является следующий – идентификатор для всех таблиц генерируется обычным триггером, выбирающим значения из генератора. При этом начальное значение генератора различно для разных БД, за счет чего обеспечится уникальность идентификатора по всем БД. Данный подход характерен для InterBase. Применение его для других СУБД может быть ограничено невозможностью задать начальное значение для счетчика автоинкрементных полей.

Недостаток данного подхода – то, что при наличии большого количества узлов снижается диапазон для каждого из них, так как диапазон INTEGER ограничен (обычно это 4-байтовое число, то есть общий диапазон составляет 4294967296 значений). Если количество узлов возрастает, то снижается диапазон адресации каждого из узлов и, соответственно, узел может принять меньшее количество своих данных.

Заключение

При использовании журнальной репликации возникает ряд побочных проблем, которые требуют разрешения для нормального функционирования распределенной базы данных.

В статье рассмотрены проблемы, которые могут возникать как в общем случае, при любой конфигурации распределенной БД, так и возникающие только при специфичных конфигурациях БД (двусторонней и многосторонней репликации).

Приведены примерные решения возникающих проблем.

ЛИТЕРАТУРА

1. *Григорьев Ю.А.*, *Плутенко А.Д.* Жизненный цикл проектирования распределенных баз данных. Благовещенск: Изд-во Амурского гос. ун-та, 1999.
2. *Плутенко А.Д., Ситников А.А.* Моделирование процесса репликаций данных в СУБД методом журнальной синхронизации // Информатика и системы управления, №1(7), 2004. С. 16-26.
3. *Дейт К.* Введение в системы баз данных, 6-е изд. М.: Диалектика, 1998.
4. Duggan Vince. Replicant Technologies,. http://www.synectics.co.za/borland/interbase/ibreplicator/papers/4128.html