3 Транзакции и управление параллельным доступом

Сервер баз данных, под управлением SQL Server, может одновременно выполнять запросы большого количества пользователей. При этом совершенно не исключается ситуация, когда несколько пользователей пытаются работать с одними и теми же данными. В самом простом случае несколько пользователей одновременно считывают один и тот же набор данных. Разумеется, при этом они никак не мешают друг другу — каждый пользователь считывает одинаковый набор данных. Проблемы начинаются, когда пользователи начинают изменять или удалять данные, с которыми работают другие пользователи. Проблемы возникают как при одновременном изменении данных несколькими пользователями, так и при чтении изменяемых данных. Трудно предсказать, какой набор данных могут получить пользователи из таблицы, если один или более пользователей в это время производит изменение строк. Кроме того, если операция изменения данных оказалась неуспешной, целостность и достоверность данных в таблице может быть нарушена. Для большинства задач подобная неоднозначность является критической.

В этой главе мы рассмотрим механизмы, которые использует SQL Server 2005 для решения задач одновременного доступа пользователей к данным. Однако сначала мы обсудим методы обеспечения целостности данных и согласованности изменений, производимых над ними. В SQL Server эти методы базируются на понятии транзакций.

Транзакции

Принципы, лежащие в основе функционирования механизма хранения данных SQL Server, по сути, аналогичны принципам функционирования всех современных СУБД. Он отвечает за организацию хранения данных, доступа к ним, а также гарантирует обеспечение целостности этих данных в процессе их модификации множеством пользователей. При этом обеспечение целостности хранимых данных предполагает выполнение четырех условий:

- В последовательности операций, производимых над данными, должны быть выполнены либо все операции, либо не должна быть выполнена ни одна. Это условие называется условием *атомарности*.

- Любая последовательность операций должна после своего завершения оставлять систему в *согласованном* (consistent) состоянии. Данное условие носит названия условия *согласованности*.
- Изменения, производимые над данными в рамках выполнения последовательности операций, не должны быть видны до тех пор, пока не закончится выполнение всей последовательности операций. По завершении выполнения последовательности операций, в случае выполнения предыдущего условия, все произведенные изменения *фиксируются* (commit) в системе хранения данных. Если выполнение последовательности операций прерывается по какой-либо причине либо завершение их выполнения оставляет систему в несогласованном состоянии, производится отмена изменений, выполненных в рамках всей последовательности. При этом принято говорить, что осуществлен *откат* (rollback) изменений. Это условие получило название условия *изолированности*.
- Зафиксированные изменения сохраняются в системе даже в случае возникновения аварийных отказов системы. Это последнее условие носит название условия *устойчивости*.

ПРИМЕЧАНИЕ

Эти четыре условия гарантированности целостности данных иногда для краткости называют ACID-требованиями. ACID – это аббревиатура от Atomicity (Атомарность), Consistency (Целостность), Isolation (Изолированность) и Durability (Устойчивость).

Рассматриваемую последовательность операций, производимых над данными, образующих единый, логически завершенный блок, который выполняется как одно целое, принято называть *транзакцией* (transaction). В транзакцию может быть включено как одна, так и несколько десятков команд. Независимо от количества команд в транзакции, либо все они будут выполнены, либо ни одна из них не выполнится. Систему, удовлетворяющую всем четырем приведенным выше условиям, принято называть *транзакциональной*.

Режимы транзакций

Как уже было замечено, под транзакцией понимается совокупность операторов и команд Transact-SQL, которые выполняются как единое целое. Чтобы реализовать это, сервер баз данных должен четко понимать, где заканчивается одна транзакция и начинается другая. В SQL Server 2005 поддерживается несколько режимов (способов) задания транзакций:

- явные транзакции;
- неявные транзакции;
- автоматически фиксируемые транзакции;
- транзакции уровня пакета.

Режим транзакций задается на уровне пользовательского соединения и фактически определяет способ, который используется данным соединением для определения границ транзакций. При этом каждое соединение может использовать свой собственный режим транзакций.

ПРИМЕЧАНИЕ

При использовании для доступа к данным механизмов OLE DB и ADO можно задействовать явное определение транзакций с помощью соответствующих методов. Если для доступа к данным используется технология ODBC, то явное определение транзакции невозможно, так как ODBC поддерживает только автоматическое и неявное определение транзакции. Для доступа к данным приложение может задействовать различные механизмы. В этой ситуации рекомендуется задействовать режимы транзакций, поддерживаемые всеми используемыми механизмами. В противном случае результат может быть непредсказуемым. Допустим, транзакция реализуется через механизм ODBC. Если приложение пытается выполнить явное начало транзакции при помощи оператора BEGIN TRANSACTION, драйвер ODBC не обработает ее соответствующим образом. В результате может произойти нарушение логики работы приложения.

Явные транзакции

Явные транзакции (explicit transaction) предполагают явное указание пользователем начала и конца транзакции при помощи соответствующих операторов Transact-SQL. Поскольку границы транзакции определяются пользователем, этот режим транзакций зачастую называют пользовательским (user-defined transaction). При явном режиме определения транзакций пользователь должен сам заботиться о соблюдении в рамках приложений логической целостности данных и бизнес-правил. Именно пользователь решает, какие операторы должны выполняться в рамках одной транзакции, а какие могут быть представлены несколькими, последовательно выполняемыми транзакциями. Ключевой задачей при планировании транзакций является обеспечение целостности данных. Операции, которые должны выполняться как единое целое (например, списание суммы с одного счета и зачисление на другой), следует объединять в одну транзакцию. И наоборот, если операции между собой логически не связаны и откат одной операции не оказывает влияние на результаты другой, необходимо выполнять их в рамках различных транзакций.

Определение начала транзакции

Каждая явная транзакция имеет свое начало и конец. Начало транзакции задается при помощи оператора `BEGIN TRANSACTION`. Этот оператор задает начало транзакции, но при этом транзакция не регистрируется в журнале до тех пор, пока не будет выполнена первая операция, изменяющая данные. Только в этот момент в журнале создается запись, отражающая начало определенной транзакции. Если в последующем транзакция будет прервана и потребуется выполнить откат, система выполнит откат всех изменений, связанных с этой транзакцией, вплоть до данной записи. Оператор имеет следующий формат:

```
BEGIN TRAN[SACTION] [<имя_транзакции>] [WITH MARK <имя_отметки>]
```

Имя транзакции может быть задано как идентификатор, так и при помощи строковой переменной. Использование переменной для указания имени транзакции позволяет нескольким пользователям создавать множество транзакций, используя один и тот же код (например, хранимую процедуру). Имя транзакции должно удовлетворять стандартным правилам именования объектов, но его длина не должна превышать 32 символа. Обычно имена транзакций используются в случае, если имеется несколько транзакций и необходимо внести ясность в работу с ними. Имена транзакций улучшают читаемость кода, однако SQL Server запоминает имя только первой транзакции и игнорирует имена всех вложенных транзакций.

Тем не менее существует возможность зарегистрировать имя транзакции в журнале. Для этого следует использовать предложение `WITH MARK`, позволяющее создать специальную именованную отметку в журнале.

ПРИМЕЧАНИЕ

Отметка будет создана только в том случае, если транзакция выполняет какие-либо изменения данных. Транзакции, осуществляющие только чтение данных, не могут порождать отметку в журнале.

В качестве имени отметки может выступать любой строковый литерал (в том числе и в формате Unicode). Длина отметки не может превышать 510 символов (255 символов, если литерал представлен в формате Unicode). Если при начале транзакции была сделана отметка в журнале, впоследствии пользователь может выполнить восстановление базы данных вплоть до указанной отметки[1].

ВНИМАНИЕ

В случае вложенных транзакций только одна из них может создавать отметку в журнале. Создание транзакции с отметкой в журнале внутри другой транзакции, также создающей отметку, не разрешается.

Сохранение состояния транзакции

Для продолжительных транзакций, состоящих из нескольких операторов, существует возможность устанавливать *точки сохранения* (savepoints). Точки сохранения используются для выполнения частичного отката транзакции. Частичный откат транзакции предполагает при выполнении некоторого условия откат всех изменений, произведенных после точки сохранения. Точки сохранения устанавливаются при помощи оператора `SAVE TRANSACTION`, имеющего следующий формат:

```
SAVE TRAN[SACTION] [<имя_точки_сохранения >]
```

[1] Возвращение базы данных в состояние на некоторый момент времени происходит путем загрузки резервных копий журнала транзакций с последующим откатом ненужных транзакций. Подробно этот процесс будет рассмотрен в главе 8.

Имя точки сохранения должно отвечать тем же правилам, что и имена транзакций (то есть длина имени не должна превышать 32 символа). Оно может быть задано как идентификатор, так и при помощи строковой переменной. В рамках транзакции можно создать несколько точек сохранения с одинаковыми именами, однако откат может быть выполнен только к самой последней из них.

ВНИМАНИЕ

Точки сохранения не могут быть созданы для распределенных транзакций. Кроме того, нельзя использовать точки сохранения в ситуации, когда соединение находится в режиме MARS и одновременно выполняется несколько пакетов Transact-SQL.

Откат явных транзакций

При необходимости пользователь может прервать выполнение транзакции и выполнить ее откат. Откат транзакции может быть вызван, например, при выполнении определенного логического условия. В ходе отката транзакции происходит откат всех произведенных в ее рамках изменений. При этом в журнале создается запись о том, что транзакция была отменена. Оператор отката транзакции имеет следующий формат:

```
ROLLBACK [TRAN[SACTION] [<имя_транзакции>|<имя_точки_сохранения>]
```

Имя транзакции может быть задано при помощи строкового литерала или строковой переменной. Аналогичным способом может быть задано и имя точки сохранения. Откат может быть произведен как для всей транзакции целиком (параметры оператора либо отсутствуют, либо указывается имя транзакции), так и только для ее части (при этом указывается имя точки сохранения).

Приложению или пользователю, инициировавшему транзакцию, в случае ее отката не передается какая-либо информация. Если же для работы приложения необходимо знать причину выполнения отката или сам факт его выполнения, следует использовать оператор `RAISERROR`.

ПРИМЕЧАНИЕ

В качестве альтернативы можно осуществить откат транзакции при помощи оператора ROLLBACK WORK. Этот оператор реализован в SQL Server по соображениям совместимости со стандартом ANSI SQL. Действие этого оператора аналогично действию, выполняемому оператором ROLLBACK TRANSACTION.

Фиксация явных транзакций

Для обозначения конца транзакции используется оператор `COMMIT TRANSACTION`. Этот оператор указывает реляционному механизму SQL Server о необходимости выполнить фиксацию транзакции в случае успешного завершения работы всех операторов, являющихся ее частью. В ходе фиксации все изменения, произведенные в рамках данной транзакции, становятся постоянными. При этом в журнале создается запись о том, что изменения зафиксированы и транзакция считается

завершенной. В дальнейшем откат этих изменений будет выполнить невозможно. Оператор фиксации транзакции имеет следующий формат:

```
COMMIT [ TRAN[SACTION] [<имя_транзакции>] ]
```

Имя транзакции может быть задано как идентификатор либо посредством строковой переменной. В любом случае указание имени транзакции является необязательным. Оно предназначено исключительно для улучшения читаемости кода. Реляционный механизм SQL Server игнорирует при фиксации указанные в операторе имена транзакций.

ПРИМЕЧАНИЕ

В качестве альтернативы можно осуществлять фиксацию транзакций при помощи оператора COMMIT WORK. Этот оператор реализован в SQL Server по соображениям совместимости со стандартом ANSI SQL. Действие этого оператора аналогично действию, выполняемому оператором COMMIT TRANSACTION.

Неявные транзакции

При работе в режиме неявного начала транзакций (implicit transaction) реляционный механизм SQL Server автоматически начинает новую транзакцию, как только завершается предыдущая. В этом режиме явно не задается начало транзакции, но должен быть явно определен момент ее завершения. Транзакция продолжается до тех пор, пока пользователь явно не укажет команду отката (оператор `ROLLBACK TRANSACTION`) или фиксации (оператор `COMMIT TRANSACTION`) транзакции, после чего сервер автоматически начинает новую транзакцию. В итоге создается непрерывная цепь транзакций.

Если для соединения устанавливается режим неявных транзакций, сервер автоматически начнет первую транзакцию, как только будет выполнен один из следующих операторов:

- `CREATE`. Создание некоторого объекта базы данных.
- `ALTER TABLE`. Изменение структуры таблицы.
- `DELETE`. Удаление строк данных из таблицы.
- `DROP`. Удаление некоторого объекта базы данных.
- `TRUNCATE TABLE`. Усечение таблицы.
- `SELECT`. Выборка данных из таблиц.
- `UPDATE`. Изменение данных в таблице.
- `INSERT`. Добавление строк в таблицу.
- `OPEN`. Открытие курсора.
- `FETCH`. Извлечение некоторых значений из курсора.
- `GRANT`. Предоставление доступа к объектам базы данных.
- `REVOKE`. Неявный отзыв доступа к объектам базы данных.

Транзакция продолжается до тех пор, пока не встретится оператор фиксации транзакции или оператор, предписывающий выполнить ее откат. После того как транзакция зафиксирована, реляционный механизм SQL Server инициирует начало новой транзакции в случае выполнения одного из перечисленных выше операторов.

Режим неявных транзакций устанавливается на уровне соединения при помощи специальной команды:

```
SET IMPLICIT_TRANSACTION ON
```

Для отмены режима неявных транзакций и возвращения к режиму автоматически фиксируемых транзакций следует выполнить команду:

```
SET IMPLICIT_TRANSACTION OFF
```

ПРИМЕЧАНИЕ

Режим неявных транзакций был реализован в SQL Server из соображений обеспечения переносимости кода, написанного для баз данных других производителей, использующих подобный режим управления транзакциями. В подавляющем большинстве ситуаций использование этого режима нежелательно, поскольку может привести к путанице.

Автоматически фиксируемые транзакции

В режиме *автоматически фиксируемых транзакций* (autocommit transaction) каждый оператор Transact-SQL рассматривается как отдельная транзакция. Пользователю при этом не требуется явно указывать начало и конец транзакции, поскольку SQL Server автоматически фиксирует или откатывает изменения, производимые каждым оператором. Если выполнение оператора заканчивается успешно, произведенные им изменения будут зафиксированы. Если же при выполнении оператора произошла ошибка, то для всех произведенных им изменений будет инициирован откат. В режиме автоматически фиксируемых транзакций приложение или пользователь всегда имеют возможность использовать режим явного задания транзакций. Для этого следует использовать операторы явного управления транзакциями. Как только встретится оператор `BEGIN TRANSACTION`, реляционный механизм переходит в режим явной транзакции. Все операторы, следующие за `BEGIN TRANSACTION`, не будут фиксироваться до тех пор, пока не будет явно предписано выполнить их фиксацию или откат. После того как транзакция будет зафиксирована или для нее будет выполнен откат, сервер вновь перейдет к режиму автоматически фиксируемых транзакций.

По умолчанию SQL Server работает в режиме автоматического начала транзакций. В дальнейшем пользователь может установить на уровне сеанса режим неявных транзакций либо явно указать начало и конец транзакции, переходя в режим явных транзакций.

Транзакции уровня пакета

Пакет (batch) представляет собой группу операторов Transact-SQL, передаваемых приложением для выполнения серверу баз данных как одно целое. Все операторы, содержащиеся в пакете, компилируются сервером баз данных как единое целое и образуют один план исполнения. В SQL Server 2005 для соединения можно установить режим *нескольких активных результирующих множеств* (Multiple Active Result Set, MARS), позволяющий приложению запускать несколько пакетов или запросов одновременно. При этом все пакеты, использующие одно соединение, имеют связанную среду исполнения по умолчанию. *Среда исполнения пакета* (batch execution environment) определяется следующими параметрами и компонентами:

- Параметры соединения, установленные при помощи команды `SET` (включая значения параметров `ANSI_NULLS`, `DATE_FORMAT`, `LANGUAGE` и `TEXTSIZE`).
- Контекст безопасности, определяющий набор полномочий и разрешений. Зависит от того, в контексте какого пользователя или роли приложения выполняется данное соединение.
- Контекст базы данных, определяемый тем, какая база данных является текущей для данного соединения.
- Ряд переменных, описывающих состояние среды исполнения, включая переменные `@@ERROR`, `@@ROWCOUNT`, `@@FETCH_STATUS` и `@@IDENTITY`.
- Временные таблицы.

Другими словами, пакеты, переданные в режиме MARS, не видят изменений среды исполнения, производимых другими пакетами. Каждый раз, когда пакет начинает выполняться, он получает то же состояние среды исполнения, что и другие пакеты. В предыдущих версиях SQL Server в один момент времени в контексте одного соединения мог выполняться только один пакет. При этом каждый последующий пакет ощущал на себе изменения среды исполнения, произведенные другими пакетами.

ПРИМЕЧАНИЕ

Может сложиться ложное впечатление об идентичности понятий пакета и транзакции. И то и другое является объединением операторов Transact-SQL в единый логический блок. Однако пакет представляет собой набор операторов, которыми манипулирует клиентское приложение, взаимодействуя с сервером баз данных. Транзакция же используется сервером для обеспечения целостности базы данных. Пакет может включать в себя несколько транзакций, так же как и транзакция может охватывать несколько пакетов.

В ситуации, когда транзакция (явная или неявная) выполняется в контексте пакета MARS, она рассматривается как транзакция уровня пакета. Если на момент завершения пакета некоторая транзакция не была зафиксирована, SQL Server выполнит для нее автоматический откат.

Вложенные транзакции

Допустимой считается ситуация, когда внутри одной транзакции запускается еще одна транзакция. В этой ситуации говорят о вложенных транзакциях. *Вложенной транзакцией* (nested transaction) называется транзакция, выполнение которой инициируется из тела активной транзакции. Использование вложенных транзакций доступно только при работе с явными транзакциями. При использовании автоматически фиксируемых или неявных транзакций перед тем, как начнется новая транзакция, предыдущая транзакция должна быть завершена.

Как правило, к вложенным транзакциям прибегают в ситуации, когда необходимо реализовать выполнение хранимой процедуры в рамках уже активной транзакции. Пользователь может обращаться к хранимой процедуре как из уже начатой транзакции, так и непосредственно (не из тела транзакции). Сервер должен гарантировать целостность данных и соблюдение требования ACID в обоих случаях.

Для создания вложенной транзакции пользователю не нужно использовать какие-либо дополнительные команды. Он просто начинает новую транзакцию, не закрыв предыдущую. Несмотря на наличие операторов `COMMIT TRANSACTION`, фиксация всех вложенных транзакций откладывается до завершения вызвавшей их транзакции. Если для вызвавшей их транзакции выполняется откат, то для всех затронутых при этом вложенных транзакций будет также выполнен откат. Фиксация всех вложенных транзакций будет выполнена при фиксации вызывающей их транзакции.

В случае вложенных транзакций команда `COMMIT TRANSACTION` всегда фиксирует последнюю начатую транзакцию, фактически завершая ее. Даже если в команде `COMMIT TRANSACTION` указано имя транзакции более высокого уровня, завершена будет транзакция, начатая последней. По-другому обрабатывается операция отката изменений. Если в операторе `ROLLBACK TRANSACTION` указывается имя вызывающей транзакции, то вместе с вызывающей транзакцией происходит откат всех вложенных транзакций. В ситуации, когда оператор `ROLLBACK TRANSACTION` используется без указания имени транзакции, откат всегда выполняется к самому внешнему незафиксированному оператору `BEGIN TRANSACTION`. При этом будет выполнен откат всех вложенных транзакций.

Для демонстрации принципа работы с вложенными транзакциями рассмотрим пример. Допустим, имеется таблица:

```
CREATE TABLE SimpleTable
       (id INT PRIMARY KEY, string NVARCHAR(20) NOT NULL)
```

Кроме того, в базе данных создана хранимая процедура, осуществляющая вставку строк в указанную таблицу:

```
CREATE PROCEDURE SimpleInsert
       (@id INT, @str NVARCHAR(20))
```

```
  AS
  BEGIN TRANSACTION
      INSERT INTO SimpleTable VALUES (@id, @str)
  COMMIT TRANSACTION
```

Обратите внимание, что вставка строки выполняется хранимой процедурой в рамках транзакции. В примере у нас в транзакции всего один оператор, в то время как на практике внутри транзакции их может быть сколько угодно много. Посмотрим, как поведет себя хранимая процедура в случае использования внутри других транзакций:

```
BEGIN TRANSACTION
  EXEC SimpleInsert 1, N'Это первая строка'
SAVE TRANSACTION point1
  EXEC SimpleInsert 2, N'Это вторая строка'
ROLLBACK TRANSACTION point1
  EXEC SimpleInsert 3, N'Это третья строка'
COMMIT TRANSACTION
EXEC SimpleInsert 4, N'Это четвертая строка'
```

Первые три вызова хранимой процедуры выполняются в рамках одной транзакции. Четвертый вызов выполняется уже после ее фиксации. Если по окончании выполнения этих операторов мы попытаемся просмотреть содержимое таблицы, мы получим следующий результат:

```
SELECT * FROM SimpleTable

id          string
----------- --------------------
1           Это первая строка
3           Это третья строка
4           Это четвертая строка

(3 row(s) affected)
```

Обратите внимание, что операторы в теле хранимой процедуры выполнялись в рамках самостоятельной транзакции. Однако, будучи вызванной из другой транзакции, транзакция в хранимой процедуре рассматривается как вложенная. Поэтому при частичном откате изменений вызвавшей ее транзакции будет выполнен откат и всех зафиксированных ей изменений.

Специальная переменная среды исполнения `@@TRANCOUNT` представляет собой счетчик активных транзакций для данного соединения. Каждый раз, когда оператор `BEGIN TRANSACTION` начинает новую транзакцию, значение этой переменной увеличивается на 1. Фиксация транзакции приводит к уменьшению значения переменной `@@TRANCOUNT` на 1. При откате транзакции при помощи оператора

ROLLBACK TRANSACTION значение @@TRANCOUNT также уменьшается на 1. В случае вложенных транзакций выполнение команды ROLLBACK TRANSACTION без указания имени транзакции приводит к откату всех транзакций и устанавливает значение переменной @@TRANCOUNT в 0.

Недопустимые в транзакциях операторы

Не каждая операция может быть выполнена в рамках транзакции. Принцип транзакциональности предполагает возможность выполнения отката изменений, производимых операторами, являющимися частью транзакции. В рамках транзакции не могут быть выполнены операции, для которых невозможно выполнить откат. Примером такой операции является создание базы данных. Для отмены изменений, вызванных действием таких операций, рекомендуется прибегать к восстановлению из резервных копий.

В транзакциях недопустимыми являются следующие операции:

- создание базы данных при помощи оператора CREATE DATABASE;
- изменение конфигурации базы данных при помощи оператора ALTER DATABASE;
- удаление базы данных при помощи оператора DROP DATABASE;
- резервное копирование при помощи оператора BACKUP;
- восстановление базы данных из резервной копии при помощи оператора RESTORE;
- активирование изменений параметров настройки сервера, произведенных системной хранимой процедурой sp_configure, при помощи оператора RECONFIGURE;
- обновление статистической информации посредством вызова оператора UPDATE STATISTICS.

Внутри транзакций запрещается изменений параметров базы данных при помощи системной хранимой процедуры sp_dboption, а также любой другой хранимой процедуры, изменяющей значения в системной базе данных master.

Распределенные транзакции

Когда пользователю необходимо обратиться к ресурсам, размещенным в разных базах данных (даже если они находятся на одном физическом сервере), он должен использовать *распределенные транзакции* (distributed transaction). На самом деле, распределенная транзакция представляет собой несколько отдельных транзакций, выполняемых локально в каждой базе данных, к которой обращается пользователь. В ситуации, когда базы данных расположены на разных серверах, каждый из них выполняет свою часть обработки данных. Если же базы данных расположены в пределах одного экземпляра, всю работу по обработке данных

выполняет данный экземпляр. Для приложения работа с распределенными транзакциями в целом не отличается от работы с локальными транзакциями, поскольку все необходимые действия по координации выполняются сервером автоматически.

Каждый сервер, вовлеченный в распределенную транзакцию, выступает в качестве *диспетчера ресурсов* (resource manager). Диспетчеры ресурсов выполняют фиксацию или откат локальных транзакций, координируя при этом свои действия с диспетчером транзакции. *Диспетчером транзакции* (transaction manager) выступает специальный программный компонент, берущий на себя обязанности по выполнению фиксации и отката распределенной транзакции в целом. Диспетчер транзакций координирует работу диспетчеров ресурсов и гарантирует, что все локальные транзакции будут зафиксированы одновременно либо для них для всех сразу будет выполнен откат. Исключается ситуация, когда изменения на одном сервере будут зафиксированы, а для изменений, произведенных на другом сервере, будет выполнен откат.

ПРИМЕЧАНИЕ

В качестве диспетчера транзакций в SQL Server используется служба Distribution Transaction Coordinator (MS DTC).

Алгоритм фиксации распределенных транзакций отличается от применяемого для фиксации транзакций, выполняющихся локально. Связано это со спецификой распределенных транзакций. Допустим, распределенная транзакция предполагает изменение данных на четырех серверах. В определенный момент времени три сервера выполняют изменение данных, фиксируют эти изменения, о чем и извещают диспетчера транзакции. В этот момент четвертый сервер сообщает диспетчеру транзакций о невозможности выполнить запрошенные изменения. В такой ситуации диспетчер должен известить три других сервера о необходимости выполнения отката произведенных изменений. Однако эти серверы уже выполнили фиксацию изменений и откат этих изменений уже невозможен. Таким образом, для фиксации распределенных транзакций указанный алгоритм не подходит. Вместо этого, при работе с распределенными транзакциями применяется алгоритм *двухфазной фиксации* (two-phased commit, 2PC). Этот алгоритм, как следует из его названия, включает две фазы:

- **Фаза подготовки (prepare phase).** Получив запрос на фиксацию распределенной транзакции, диспетчер транзакции (в роли которого выступает служба DTC) отправляет всем задействованным диспетчерам ресурсов предписание выполнить фиксацию локальных изменений. Каждый из диспетчеров ресурсов предпринимает попытку сброса измененных страниц данных на диск, не выполняя при этом фиксацию изменений в журнале. По результатам этих действий диспетчеры ресурсов сообщают диспетчеру транзакций об успешности завершения локальной транзакции либо о необходимости произведения отката.

- **Фаза фиксации (commit phase).** Только получив от всех диспетчеров ресурсов подтверждение готовности к фиксации транзакций, диспетчер транзакций отдает команду выполнить эту фиксацию. Если один или более диспетчеров ресурсов информирует о необходимости выполнения отката, служба DTC отменяет фиксацию транзакции и инициирует процесс отката распределенной транзакции. При этом всем диспетчерам ресурсов отправляется требование отката транзакций.

Распределенная транзакция может быть начата несколькими способами:

- Начало распределенной транзакции может быть явно задано при помощи оператора `BEGIN DISTRIBUTED TRANSACTION`.
- В рамках локальной транзакции используется распределенный запрос. Например, может осуществляться обновление данных в секционированных таблицах. В этом случае сервер автоматически преобразовывает локальную транзакцию в распределенную.
- В случае если параметр соединения `REMOTE_PROC_TRANSACTION` равен `ON`, разрешается вызывать хранимую процедуру, расположенную на *удаленном сервере* (remote server). Данный параметр оставлен в SQL Server 2005 по соображениям совместимости. Тем не менее, если при указанном значении этого параметра приложение начинает локальную транзакцию и из нее вызывает удаленную хранимую процедуру, данная транзакция будет автоматически расширена до распределенной транзакции.
- Приложение может начать распределенную транзакцию, используя методы OLE DB или функции ODBC.

ПРИМЕЧАНИЕ

Хотя для явного задания начала распределенной транзакции используется специальный оператор BEGIN DISTRIBUTED TRANSACTION, фиксация или откат распределенной транзакции осуществляется все теми же операторами COMMIT TRANSACTION и ROLLBACK TRANSACTION, что были рассмотрены ранее в этой главе.

Управление параллельным доступом

Как уже было замечено ранее, изменения данных, выполняемые одной транзакцией, не должны зависеть от изменений, выполняемых другой транзакцией, то есть изменения данных различными транзакциями должны быть *изолированными*. Стандарт ANSI/ISO SQL92 определяет четыре уровня изолированности транзакций, характеризующиеся разной степенью независимости этих транзакций друг от друга. По сути, уровни изолированности определяют схему обработки параллельно выполняющихся транзакций. Одни и те же транзакции, выполненные на разных уровнях изолированности над одним и тем же массивом данных, могут дать существенным образом различающиеся результаты.