Транзакции

Транзакция — это последовательность операций, выполняемая как единое целое. В составе транзакций можно исполнять почти все операторы языка Transact-SQL. Если при выполнении транзакции не возникает никаких ошибок, то все модификации базы данных, сделанные во время выполнения транзакции, становятся постоянными. Транзакция выполняется по принципу «все или ничего». Транзакция не оставляет данные в промежуточном состоянии, в котором база данных не согласована. Транзакция переводит базу данных из одного целостного состояния в другое.

В качестве примера транзакции рассмотрим последовательность операций по приему заказа в коммерческой компании. Для приема заказа от клиента приложение ввода заказов должно:

- выполнить запрос к таблице товаров и проверить наличие товара на складе;
- добавить заказ к таблице счетов;
- обновить таблицу товаров, вычтя заказанное количество товаров из количества товара, имеющегося в наличии;
- обновить таблицу продаж, добавив стоимость заказа к объему продаж служащего, принявшего заказ;
- обновить таблицу офисов, добавив стоимость заказа к объему продаж офиса, в котором работает данный служащий.

Для поддержания целостности транзакция должна обладать четырьмя свойствами АСИД: атомарность, согласованность, изоляция и долговечность. Эти свойства называются также ACID-свойствами (от англ., atomicity, consistency, isolation, durability).

- Атомарность (Atomicity). Транзакция должна представлять собой атомарную (неделимую) единицу работы (исполняются либо все модификации, из которых состоит транзакция, либо ни одна).
- Согласованность (или Непротиворечивость) (Consistency). По завершении транзакции все данные должны
  остаться в согласованном состоянии. Чтобы сохранить целостность всех данных, необходимо выполнение модификации транзакций по всем правилам, определенным в реляционных СУБД.
- Изоляция (Isolation). Модификации, выполняемые одними транзакциями, следует изолировать от модификаций, выполняемых другими транзакциями параллельно. Уровни изоляции транзакции могут изменяться в широких пределах. На каждом уровне изоляции достигается определенный компромисс между степенью распараллеливания и степенью непротиворечивости. Чем выше уровень изоляции, тем выше степень непротиворечивости данных. Но чем выше степень непротиворечивости, тем ниже степень распараллеливания и тем ниже степень доступности данных.
- Долговечность (или Устойчивость) (Durability). По завершении транзакции ее результат должен сохраниться в системе, несмотря на сбой системы, либо (что касается незафиксированных транзакций) может быть полностью отменен вслед за сбоем системы.

Способы определения границ транзакций

Замечания:

- В данной теме не будут учитываться особенности режима MARS (Multiple Active Result Sets). Режим MARS — это функция, которая появилась в SQL Server 2005 и ADO.NET 2.0 для выполнения нескольких пакетов по одному соединению. По умолчанию режим MARS отключен. Включить его можно, добавив в строку соединения параметр «MultipleActiveResultSets=True».

- В данной теме не будут рассматриваться распределенные транзакции.

По признаку определения границ различают автоматические, неявные и явные транзакции.

Автоматические транзакции. Режим автоматической фиксации транзакций является режимом управления транзакциями SQL Server по умолчанию. В этом режиме каждая инструкция T-SQL выполняется как отдельная транзакция. Если выполнение инструкции завершается успешно, происходит фиксация; в противном случае происходит откат. Если возникает ошибка компиляции, то план выполнения пакета не строится и пакет не выполняется.

В следующем примере ни одна из инструкций INSERT во втором пакете не выполнится из-за ошибки компиляции. При этом произойдет откат первых двух инструкций INSERT, и они не будут выполняться:

```
CREATE TABLE Tab1 (
    Col1 int NOT NULL PRIMARY KEY,
    Col2 char(3)
);

GO
INSERT INTO Tab1 VALUES (1, 'aaa');
INSERT INTO Tab1 VALUES (2, 'bbb');
INSERT INTO Tab1 VALUSE (3, 'ccc'); -- Синтаксическая ошибка.

GO
SELECT * FROM Tab1; -- Результат пустой. GO
```

В следующем примере третья инструкция INSERT вызывает ошибку дублирования первичного ключа во время выполнения. Поэтому первые две инструкции INSERT выполняются успешно и фиксируются, а третья инструкция INSERT вызывает ошибку времени выполнения и не выполняется.

```
CREATE TABLE Tab1 (
    Col1 int NOT NULL PRIMARY KEY,
    Col2 char(3)
);

GO
INSERT INTO Tab1 VALUES (1, 'aaa');
INSERT INTO Tab1 VALUES (2, 'bbb');
INSERT INTO Tab1 VALUES (1, 'ccc'); -- Ошибка времени исполнения. GO
SELECT * FROM Tab1; -- Возвращаются две строки.
GO
```

Неявные транзакции. Если соединение работает в режиме неявных транзакций, то после фиксации или отката текущей транзакции SQL Server автоматически начинает новую транзакцию. В этом режиме явно указывается только граница окончания транзакции с помощью инструкций COMMIT TRANSACTION и ROLLBACK TRANSACTION. Для ввода в действие поддержки неявных транзакций применяется инструкция SET IMPLICIT_TRANSACTION ON. В конце каждого пакета необходимо отключать этот режим. По умолчанию режим неявных транзакций в SQL Server отключен.

В следующем примере сначала создается таблица Tab1, затем включается режим неявных транзакций, после чего начинаются две транзакции. После их исполнения режим неявных транзакций отключается:

```
CREATE TABLE Tab1 (
    Col1 int NOT NULL PRIMARY KEY,
    Col2 char(3) NOT NULL
)

GO
SET IMPLICIT_TRANSACTIONS ON
-- Первая неявная транзакция, начатая оператором INSERT INSERT INTO Tab1 VALUES (1,
'aaa')
INSERT INTO Tab1 VALUES (2, 'bbb')
COMMIT TRANSACTION -- Фиксация первой транзакции
-- Вторая неявная транзакция, начатая оператором INSERT INSERT INTO Tab1 VALUES (3,
'ccc')
SELECT * FROM Tab1
COMMIT TRANSACTION -- Фиксация второй транзакции
SET IMPLICIT_TRANSACTIONS OFF
GO
```

Явные транзакции. Для определения явных транзакций используются следующие инструкции:

- BEGIN TRANSACTION – задает начальную точку явной транзакции для соединения;
- COMMIT TRANSACTION или COMMIT WORK – используется для успешного завершения транзакции, если не возникла ошибка;
- ROLLBACK TRANSACTION или ROLLBACK WORK – используется для отмены транзакции, во время которой возникла ошибка.
- SAVE TRANSACTION – используется для установки точки сохранения или маркера внутри транзакции. Точка сохранения определяет место, к которому может возвратиться транзакция, если часть транзакции условно отменена. Если транзакция откатывается к точке сохранения, то ее выполнение должно быть продолжено до завершения с обработкой дополнительных инструкций языка T-SQL, если необходимо, и инструкции COMMIT TRANSACTION, либо транзакция должна быть полностью отменена откатом к началу. Для отмены всей транзакции следует использовать инструкцию ROLLBACK TRANSACTION; в этом случае отменяются все инструкции транзакции.

В следующем примере демонстрируется использование точки сохранения транзакции для отката изменений:

```
USE pubs
GO

BEGIN TRANSACTION royaltychange
        UPDATE titleauthor
        SET royaltyper = 65
        FROM titleauthor, titles
        WHERE royaltyper = 75 AND titleauthor.title_id = titles.title_id
                AND title = 'The Gourmet Microwave'
```

```
        UPDATE titleauthor
        SET royaltyper = 35
        FROM titleauthor, titles
        WHERE royaltyper = 25 AND titleauthor.title_id = titles.title_id
                AND title = 'The Gourmet Microwave'
SAVE TRANSACTION percentchanged

/* После того, как обновлено royaltyper для двух авторов, вставляется точка сохранения percentchanged, а затем определяется, насколько изменится заработок авторов после увеличения на 10 процентов цены книги */
UPDATE titles
SET price = price * 1.1
WHERE title = 'The Gourmet Microwave'

SELECT (price * royalty * ytd_sales) * royaltyper
FROM titles, titleauthor
WHERE title = 'The Gourmet Microwave' AND titles.title_id = titleauthor.title_id

/* Откат транзакции до точки сохранения и фиксация транзакции в целом */
ROLLBACK TRANSACTION percentchanged
COMMIT TRANSACTION
```

Режим явных транзакций действует только на протяжении данной транзакции. После завершения явной транзакции соединение возвращается в режим, заданный до запуска этого режима, то есть в неявный или автоматический.

Функции для обработки транзакций

- @@TRANCOUNT возвращает число активных транзакций для текущего соединения. Инструкция BEGIN TRANSACTION увеличивает значение @@TRANCOUNT на 1, а инструкция ROLLBACK TRANSACTION уменьшает его до 0 (исключение — инструкция ROLLBACK TRANSACTION имя_точки_сохранения, которая не влияет на значение @@TRANCOUNT). Инструкции COMMIT TRANSACTION уменьшают значение @@TRANCOUNT на 1.
- XACT_STATE () сообщает о состоянии пользовательской транзакции текущего выполняемого запроса в соответствии с данными, представленными в таблице.

Возвращаемое значение	Пояснение
1	Текущий запрос содержит активную пользовательскую транзакцию и может выполнять любые действия, включая запись данных и фиксацию транзакции.
0	У текущего запроса нет активной пользовательской транзакции.
-1	В текущем запросе есть активная транзакция, однако произошла ошибка, из-за которой транзакция классифицируется как не фиксируемая. Запросу не удается зафиксировать транзакцию или выполнить откат до точки сохранения; можно только запросить полный откат транзакции.

Ограничения.

- Функция @@TRANCOUNT не может использоваться для определения фиксируемости транзакции.
- Функция XACT_STATE не может использоваться для определения наличия вложенных транзакций.

Ошибки, возникающие в процессе обработки транзакций

Если ошибка делает невозможным успешное выполнение транзакции, SQL Server автоматически выполняет ее откат и освобождает ресурсы, удерживаемые транзакцией. Если сетевое соединение клиента с SQL Server разорвано, то после того, как SQL Server получит уведомление от сети о разрыве соединения, выполняется откат всех необработанных транзакций для этого соединения. В случае сбоя клиентского приложения, выключения либо перезапуска клиентского компьютера соединение также будет разорвано, а SQL Server выполнит откат всех необработанных транзакций после получения уведомления о разрыве от сети. Если клиент выйдет из приложения, выполняется откат всех необработанных транзакций.

В случае ошибки (нарушения ограничения целостности) во время выполнения инструкции в пакете по умолчанию SQL Server выполнит откат только той инструкции, которая привела к ошибке. Это поведение можно изменить с помощью инструкции SET XACT_ABORT. После выполнения инструкции SET XACT_ABORT ON любая ошибка во время выполнения инструкции приведет к автоматическому откату текущей транзакции.

На случай возникновения ошибок код приложения должен содержать исправляющее действие: COMMIT или ROLLBACK. Эффективным средством для обработки ошибок, включая ошибки транзакций, является конструкция языка Transact-SQL TRY...CATCH.

Пример

```
USE MyDB;
GO
IF OBJECT_ID ( N'dbo.SaveTranExample', N'P' ) IS NOT NULL DROP PROCEDURE dbo.SaveTranExample;
GO

CREATE PROCEDURE dbo.SaveTranExample ... -- Список параметров AS
--  Надо проверить, была ли процедура вызвана из активной транзакции, и если да,
--  то устанавить точку сохранения для последующего использования.
--  @TranCounter = 0 означает, что активной транзакции нет и процедура явно начинает
--  локальную транзакцию.
--   @TranCounter > 0 означает, что активная транзакция началась еще до вызова процедуры.
DECLARE @TranCounter INT;
SET @TranCounter = @@TRANCOUNT;
IF @TranCounter > 0
        SAVE TRANSACTION ProcedureSave;
ELSE
        BEGIN TRANSACTION;
-- Пытаемся модифицировать базу данных.
BEGIN TRY
-- INSERT, UPDATE, DELETE
```

```
...
-- Здесь мы окажемся, если не произойдет никакой ошибки.
-- Если транзакция началась внутри процедуры, то выполнить COMMIT TRANSACTION.
-- Если транзакция началась до вызова процедуры, то выполнять COMMIT TRANSACTION нельзя.
IF @TranCounter = 0
-- @TranCounter = 0 означает, что никакая транзакция до вызова процедуры не начиналась. -- Процедура должна завершить начатую в ней транзакцию.
        COMMIT TRANSACTION;
END TRY
BEGIN CATCH
-- Здесь мы окажемся, если произошла ошибка.
-- Надо определить, какой тип отката транзакции применять.
IF @TranCounter = 0
-- Транзакция началась в теле процедуры. -- Полный откат транзакции.
        ROLLBACK TRANSACTION;
            ELSE

-- Транзакция началась еще до вызова процедуры.
-- Нельзя отменять изменения, сделанные до вызова процедуры.
IF XACT_STATE() <> -1
-- Если транзакция активна, то выполнить откат до точки сохранения.
ROLLBACK TRANSACTION ProcedureSave;
-- Если транзакция актива, однако произошла ошибка, из-за которой транзакция
-- классифицируется как нефиксируемая,
-- то вернуться в точку вызова процедуры, где должен произойти откат внешней транзакции.
-- В точку вызова процедуры передается информация об ошибках.
DECLARE @ErrorMessage NVARCHAR(4000); DECLARE @ErrorSeverity INT;
DECLARE @ErrorState INT;
SELECT @ErrorMessage = ERROR_MESSAGE(); SELECT @ErrorSeverity = ERROR_SEVERITY(); SELECT @ErrorState = ERROR_STATE();
RAISERROR(@ErrorMessage, @ErrorSeverity, @ErrorState); END CATCH
GO
```

Пояснения к примеру

Для получения сведений об ошибках в области блока CATCH конструкции TRY...CATCH можно использовать следующие системные функции:

- ERROR_LINE() - возвращает номер строки, в которой произошла ошибка.
- ERROR_MESSAGE() - возвращает текст сообщения, которое будет возвращено приложению. Текст содержит значения таких подставляемых параметров, как длина, имена объектов или время.
- ERROR_NUMBER() - возвращает номер ошибки.
- ERROR_PROCEDURE() - возвращает имя хранимой процедуры или триггера, в котором произошла ошибка. Эта функция возвращает значение NULL, если данная ошибка не была совершена внутри хранимой процедуры или триггера.
- ERROR_SEVERITY() - возвращает уровень серьезности ошибки.
- ERROR_STATE() - возвращает состояние.

Инструкция RAISERROR позволяет вернуть приложению сообщение в формате системных ошибок или предупреждений.

Управлением параллельным выполнением транзакций

Когда множество пользователей одновременно пытаются модифицировать данные в базе данных, необходимо создать систему управления, которая защитила бы модификации, выполненные одним пользователем, от негативного воздействия модификаций, сделанных другими. Выделяют два типа управления параллельным выполнением:

- **Пессимистическое** управление параллельным выполнением.
- **Оптимистическое** управление параллельным выполнением.

Пессимистическое управление реализуется с помощью технологии блокировок, оптимистическое управления реализуется с помощью технологии версии строк.

Система блокировок не разрешает пользователям выполнить модификации, влияющие на других пользователей. Если пользователь выполнил какое-либо действие, в результате которого установлена блокировка, то другим пользователям не удастся выполнять действия, конфликтующие с установленной блокировкой, пока владелец не освободит ее. Пессимистическое управление используется главным образом в средах, где высока конкуренция за данные.

В случае управления на основе версий строк пользователи не блокируют данные при чтении. Во время обновления система следит, не изменил ли другой пользователь данные после их прочтения. Если другой пользователь модифицировал данные, генерируется ошибка. Как правило, получивший ошибку пользователь откатывает транзакцию и повторяет операцию снова. Этот способ управления в основном используется в средах с низкой конкуренцией за данные.

SQL Server поддерживает различные механизмы оптимистического и пессимистического управления параллельным выполнением. Пользователю предоставляется право определить тип управления параллельным выполнением, установив уровень изоляции транзакции для соединения и параметры параллельного выполнения для курсоров.

Если в случае пессимистического управления в СУБД не реализованы механизмы блокирования, то при одновременном чтении и изменении одних и тех же данных несколькими пользователями могут возникнуть следующие проблемы одновременного доступа:

проблема последнего изменения возникает, когда несколько пользователей изменяют одну и ту же строку, основываясь на ее начальном значении; тогда часть данных будет потеряна, т.к. каждая последующая транзакция перезапишет изменения, сделанные предыдущей. Выход из этой ситуации заключается в последовательном внесении изменений;

проблема "грязного" чтения (Dirty Read) возможна в том случае, если пользователь выполняет сложные операции обработки данных, требующие множественного изменения данных перед тем, как они обретут логически верное состояние. Если во время изменения данных другой пользователь будет считывать их, то может оказаться, что он получит логически неверную информацию. Для исключения подобных проблем необходимо производить считывание данных после окончания всех изменений;

проблема неповторяемого чтения (Non-repeatable or Fuzzy Read) является следствием неоднократного считывания транзакцией одних и тех же данных. Во время выполнения первой транзакции другая может внести в данные изменения, поэтому при повторном чтении первая транзакция получит уже иной набор данных, что приводит к нарушению их целостности или логической несогласованности;

проблема чтения фантомов (Phantom) появляется после того, как одна транзакция выбирает данные из таблицы, а другая вставляет или удаляет строки до завершения первой. Выбранные из таблицы значения будут некорректны.

Для решения перечисленных проблем в стандарте ANSI SQL определены четыре уровня блокирования. Уровень изоляции транзакции определяет, могут ли другие (конкурирующие) транзакции вносить изменения в данные, измененные текущей транзакцией, а также может ли текущая транзакция видеть изменения, произведенные конкурирующими транзакциями, и наоборот. Каждый последующий уровень поддерживает требования предыдущего и налагает дополнительные ограничения:

уровень 0 – запрещение «загрязнения» данных. Этот уровень требует, чтобы изменять данные могла только одна транзакция; если другой транзакции необходимо изменить те же данные, она должна ожидать завершения первой транзакции;

уровень 1 – запрещение «грязного» чтения. Если транзакция начала изменение данных, то никакая другая транзакция не сможет прочитать их до завершения первой;

уровень 2 – запрещение неповторяемого чтения. Если транзакция считывает данные, то никакая другая транзакция не сможет их изменить. Таким образом, при повторном чтении они будут находиться в первоначальном состоянии;

уровень 3 – запрещение фантомов. Если транзакция обращается к данным, то никакая другая транзакция не сможет добавить новые или удалить имеющие строки, которые могут быть считаны при выполнении транзакции. Реализация этого уровня блокирования выполняется путем использования блокировок диапазона ключей. Подобная блокировка накладывается не на конкретные строки таблицы, а на строки, удовлетворяющие определенному логическому условию.

Проблемы, связанные с параллельным выполнением транзакций, разрешают, используя уровни изоляции транзакции. От уровня изоляции зависит то, в какой степени транзакция влияет на другие транзакции и испытывает влияние других транзакций. Более низкий уровень изоляции увеличивает возможность параллельного выполнения, но за это приходится расплачиваться согласованностью данных. Напротив, более высокий уровень изоляции гарантирует согласованность данных, но при этом страдает параллельное выполнение.

Стандарт ISO определяет следующие уровни изоляции:

- **read uncommitted** (самый низкий уровень, при котором транзакции изолируются до такой степени, чтобы только уберечь от считывания физически поврежденных данных);
- **read committed** (уровень по умолчанию);
- изоляция повторяющегося чтения **repeatable read**;
- изоляция упорядочиваемых транзакций **serializable** (самый высокий уровень, при котором транзакции полностью изолированы друг от друга).

SQL Server также поддерживает еще два уровня изоляции транзакций, использующих управление версиями строк.

- **read committed** с использованием управления версиями строк;
- уровень изоляции моментальных снимков **snapshot**.

Следующая таблица показывает побочные эффекты параллелизма, допускаемые различными уровнями изоляции.

Уровень изоляции	**«Грязное» чтение**	**Неповторяющееся чтение**	**Фантомное чтение**
read uncommitted	Да	Да	Да
read committed	Нет	Да	Да
repeatable read	Нет	Нет	Да
snapshot	Нет	Нет	Нет
serializable	Нет	Нет	Нет

Для установки уровня изоляции используется следующая инструкция SET TRANSACTION ISOLATION LEVEL, имеющая следующий синтаксис:

```
SET TRANSACTION ISOLATION LEVEL
        { READ UNCOMMITTED | READ COMMITTED | REPEATABLE READ | SNAPSHOT |
          SERIALIZABLE }
[ ;]
```

Описание аргументов

READ UNCOMMITTED - указывает, что инструкции могут считывать строки, которые были изменены другими транзакциями, но еще не были зафиксированы.

READ COMMITTED - указывает, что инструкции не могут считывать данные, которые были изменены другими транзакциями, но еще не были зафиксированы. Это предотвращает чтение «грязных» данных. Данные могут быть изменены другими транзакциями между отдельными инструкциями в текущей транзакции, результатом чего будет неповторяемое чтение или недействительные данные.

Напоминание. Поведение READ COMMITTED зависит от настройки аргумента базы данных READ_COMMITTED_SNAPSHOT (находится в состоянии OFF по умолчанию). REPEATABLE READ - указывает на то, что инструкции не могут считывать данные, которые были изменены, но еще не зафиксированы другими транзакциями, а также на то, что другие транзакции не могут изменять данные, читаемые текущей транзакцией, до ее завершения.

SNAPSHOT - указывает на то, что данные, считанные любой инструкцией транзакции, будут согласованы на уровне транзакции с версией данных, существовавших в ее начале. Транзакция распознает только те изменения, которые были зафиксированы до ее начала. Инструкции, выполняемые текущей транзакцией, не видят изменений данных, произведенных другими транзакциями после запуска текущей транзакции. Таким образом достигается эффект получения инструкциями в транзакции моментального снимка зафиксированных данных на момент запуска транзакции. Перед запуском транзакции, использующей уровень изоляции моментальных снимков, необходимо установить параметр базы данных ALLOW_SNAPSHOT_ISOLATION в ON. Если транзакция с уровнем изоляции моментального снимка обращается к данным из нескольких баз данных, аргумент ALLOW_SNAPSHOT_ISOLATION должен быть включен в каждой базе данных.

SERIALIZABLE - указывает следующее:

- Инструкции не могут считывать данные, которые были изменены другими транзакциями, но еще не были зафиксированы.
- Другие транзакции не могут изменять данные, считываемые текущей транзакцией, до ее завершения.
- Другие транзакции не могут вставлять новые строки со значениями ключа, которые входят в диапазон ключей, считываемых инструкциями текущей транзакции, до ее завершения.

Замечания.

- Одновременно может быть установлен только один параметр уровня изоляции, который продолжает действовать для текущего соединения до тех пор, пока не будет явно изменен.
- Уровни изоляции транзакции определяют тип блокировки, применяемый к операциям считывания.
- В любой момент транзакции можно переключиться с одного уровня изоляции на другой, однако есть одно исключение. Это смена уровня изоляции на уровень изоляции SNAPSHOT. Такая смена приводит к ошибке и откату транзакции. Однако для транзакции, которая была начата с уровнем изоляции SNAPSHOT, можно установить любой другой уровень изоляции.
- Когда для транзакции изменяется уровень изоляции, ресурсы, которые считываются после изменения, защищаются в соответствии с правилами нового уровня. Ресурсы, которые считываются до изменения, остаются защищенными в соответствии с правилами предыдущего уровня. Например, если для транзакции уровень изоляции изменяется с READ COMMITTED на SERIALIZABLE, то совмещаемые блокировки, полученные после изменения, будут удерживаться до завершения транзакции.
- Если инструкция SET TRANSACTION ISOLATION LEVEL использовалась в хранимой процедуре или триггере, то при возврате управления из них уровень изоляции будет изменен на тот, который действовал на момент их вызова. Например, если уровень изоляции REPEATABLE READ устанавливается в пакете, а пакет затем вызывает хранимую процедуру, которая меняет уровень изоляции на SERIALIZABLE, при возвращении хранимой процедурой управления пакету, настройки уровня изоляции меняются назад на REPEATABLE READ.
- Определяемые пользователем функции и типы данных среды CLR не могут выполнять инструкцию SET TRANSACTION ISOLATION LEVEL. Однако уровень изоляции можно переопределить с помощью табличной подсказки.