### Лекция 18:

* Транзакции и обработка ошибок.

**Обработка ошибок и транзакции**

**Обработка ошибок (структурированная обработка):**

Здесь мы посмотрим только на структурированную обработку ошибок, неструктурированную можете посмотреть сами, например, по ключевым словам @@error, xact\_abort.

Для обработки ошибок (как и в других языках программирования) используется блок TRY| CATCH. Конструкция, на которой нужно отлавливать возможные ошибки помещается в блок TRY. Если произошла ошибка, то управление сразу же передается блоку CATCH. Если в блоке TRY ошибки не произошло, то блок CATCH пропускается.

Пример:

BEGIN TRY

PRINT 16/2;

PRINT 'No error';

END TRY

BEGIN CATCH

PRINT 'Error';

END CATCH;

Если делить на 2, то будет:

8

No error

Если заменить на 0, то будет:

Error

В Transact-SQL каждый блок TRY связан только с одним блоком CATCH.

Есть спец-функции, которые выдают: номер ошибки, текст ошибки, номер строки, где произошла ошибка, имя процедуры, в которой произошла ошибка. С 2012 SQL всё это делает одна команда THROW. Обработку ошибок полностью можно поместить в отдельную процедуру, которую вызывать в блоке catch.

Если ошибка произошла в блоке Try, то никакое сообщение (от SQL server) клиенту не отправляется. Управление всегда передается блоку catch.

Функции ошибок:

ERROR\_NUMBER() – возвращает номер ошибки.

ERROR\_MESSAGE() – полный текст сообщения об ошибке. Текст содержит значения подставляемых параметров, таких как длина, имена объектов или время.

ERROR\_SEVERITY() – уровень серьезности ошибки.

ERROR\_STATE() – код состояния ошибки.

ERROR\_LINE() – номер строки, которая вызвала ошибку, внутри подпрограммы.

ERROR\_PROCEDURE() – имя хранимой процедуры или триггера, в котором произошла ошибка.

Ошибки с уровнем серьезности, большим или равным 20, заставляющие компонент Database Engine закрыть соединение, не могут быть обработаны конструкцией TRY…CATCH. *Если соединение не закрылось, ошибка будет обработана конструкцией TRY…CATCH.*

Ошибки с уровнем серьезности, меньшим или равным 10, рассматриваются как предупреждения или информационные сообщения и не обрабатываются с помощью конструкции TRY…CATCH.

**Транзакции:**

Транзакция – это группа последовательных операций с БД, обычно представляющих собой один логический блок команд, который может включать в себя множество действий по манипулированию/определению данных.

Пример действия, которые следует объединить в одну транзакцию для сохранения целостности данных:

Действие по переводу денег с одного счета на другой: Вы списываете деньги с одного счета и зачисляете их на другой счет. Согласитесь, что будет печально, если деньги со счета спишутся, а на второй счет не поступят по причине какой-нибудь ошибки.

Где еще важны ACID свойства транзакций: Вы вставляете новый заказ в таблицы: заказ и заказ\_данные отдельными запросами. Чтобы при вставке, например, не «испортился» номер заказа, или товары не успели зарезервироваться кем-нибудь еще, либо между созданием заказа и заказа\_данные у Вас случилась, например, дисковая ошибка, необходимо производить действия в одной транзакции. Про уровни изоляции: как учитывать многопользовательский доступ – почитайте в предложенных книжках. Кратко можно сказать следующее: пока транзакция не закончила изменять строчки, другая транзакция не сможет менять эти же строки, но может ли она их читать – зависит от уровня изоляции. Кроме того, возможна взаимная блокировка транзакций.

Что нужно учитывать при написании транзакций: как реализовать многопользовательский доступ, как обработать различные ошибки.

Как учитывать многопользовательский доступ?

* Пока транзакция не закончила изменять строчки, другая транзакция не сможет менять эти же строки
* В зависимости от уровня изоляции вторая транзакция может или не может читать изменяемые строки
* Возможна взаимная блокировка транзакций

Транзакции можно определить явно, неявно, также есть автоматический режим. Например, транзакцией является каждый отдельный запрос в Вашем скрипте, в этом случае транзакция неявная (SQL Server подтверждает автоматически транзакцию после выполнения).

Если Вы хотите явно определить транзакцию, то используйте BEGIN TRAN и COMMIT TRAN или ROLLBACK TRAN (откатка транзакции либо полностью, либо к save point).

SAVEPOINT – создает точку останова. Точка останова – промежуточный участок в транзакции, на который можно откатиться в случае необходимости:

* Позволяет дробить транзакцию на части
* Позволяет реализовать «вложенные» транзакции

В целом, что такое свойства ACID:

A (atomicity) – атомарность: каждая транзакция – атомарная единица работы. Либо выполняется всё, либо ничего.

C (consistency) – согласованность: каждая транзакция – завершившаяся или прерванная, оставляет БД в согласованном состоянии, как определяется всеми ограничениями объектов и базы данных. В этом случае при прерывании нужен откат.

I (isolation) – изоляция: каждая транзакция выполняется так, как будто она существует в изоляции от остальных транзакций по отношению к изменениям в БД. Степень изолированности может меняться в зависимости от уровня изоляции.

D (durability) – устойчивость. Каждая транзакция претерпевает прерывание сервиса.

Atomicity (атомарность):

* Никакая транзакция не будет зафиксирована в системе частично
* Выполнены либо все подоперации, либо никакая
* На практике одновременное и атомарное выполнение транзакций невозможно
* На практике «атомарность» реализуется с использованием «отката» (rollback)
* В процессе отката отменяются все уже произведенные операции

Consistency (согласованность):

* Фиксируются только допустимые результаты операций
* Согласованность применяется не только в контексте БД. *Пример: банковские транзакции*
* В ходе выполнения операции согласованность не требуется
* Вследствие атомарности промежуточная несогласованность остается скрытой

Isolation (изолированность):

* Параллельные транзакции не оказывают влияния на друг друга
* В реальных БД полная изолированность не поддерживается
* Уровень изолированности – характеристика соответствия БД свойству изолированности

Durability (устойчивость):

* Вне зависимости от сбоев системы результаты успешных транзакций сохранятся в системе
* Если пользователь получил подтверждение об успешности транзакции, гарантируется сохранность результатов

Проблемы поддержки изолированности

* Потерянное обновление: *Изменение одного блока данных несколькими транзакциями*
* «Грязное» чтение: *Чтение данных, измененных впоследствии откатившейся транзакцией*
* Неповторяющееся чтение: *Повторное чтение измененных данных одной и той же транзакцией*
* Чтение «фантомов»: *Взаимосвязанные критерии изменения данных двумя транзакциями*

Изолированность: потерянное обновление

|  |  |
| --- | --- |
| **Транзакция 1** | **Транзакция 2** |
| UPDATE table\_1  SET attr\_2 = attr\_2 + 20  WHERE attr\_1 = 1; | UPDATE table\_1  SET attr\_2 = attr\_2 + 25  WHERE attr\_1 = 1; |

Что получим в итоге?

1. attr\_2 = attr\_2 + 20

2. attr\_2 = attr\_2 + 25

3. attr\_2 = attr\_2 + 45

Результат однозначно не определен

*При одновременной попытке изменить одни и те же данных в параллельных транзакциях, одно из изменений может потеряться*

Пример: Два редактора делают электронные копии одного и того же документа. Каждый редактор меняет копию независимо, после чего сохраняет свой вариант, тем самым переписывая оригинальный документ. Редактор, который изменил копию последним, затирает изменения, внесенные другим редактором. Эту проблему можно избежать если один редактор не будет иметь доступ к файлу, пока второй не завершит транзакцию и не подтвердит ее.

Изолированность: «грязное» чтение

|  |  |
| --- | --- |
| **Транзакция 1** | **Транзакция 2** |
| UPDATE table\_1  SET attr\_2 = attr\_2 + 20  WHERE attr\_1 = 1; |  |
|  | SELECT attr\_2  FROM table\_1  WHERE attr\_1 = 1; |
| ROLLBACK WORK; |  |

Могут быть считаны данные, измененные параллельно выполняемой транзакцией, которая затем была отменена.

Пример: Редактор вносит изменения в электронный документ. Во время изменений, второй редактор делает копию документа, который включает все изменения на данный момент, распределяет документ в соответствующее место хранения. Первый редактор решает, что сделанные изменения являются ошибочными, удаляет изменения и сохраняет документ. Распределенный документ содержит изменения, которые больше не существуют и должны быть обработаны, как те, которые никогда не существовали. Эту проблему можно избежать, если никто не будет иметь доступ к чтению файла пока текущий редактор не сделает финальное сохранение и не подтвердит транзакцию.

Изолированность: неповторяющееся чтение

|  |  |
| --- | --- |
| **Транзакция 1** | **Транзакция 2** |
|  | SELECT attr\_2  FROM table\_1  WHERE attr\_1 = 1; |
| UPDATE table\_1  SET attr\_2 = attr\_2 + 20  WHERE attr\_1 = 1; |  |
| COMMIT; |  |
|  | SELECT attr\_2  FROM table\_1  WHERE attr\_1 = 1; |

При повторном прочтении в рамках одной транзакции данные оказываются измененные или удаленными транзакцией, выполнившейся параллельно.

Пример: Редактор читает один и тот же документ дважды, но между прочтениями автор переписывает текст. Когда редактор читает документ второй раз, он содержит уже измененный текст. Прочитать второй раз первоначальный текст невозможно. Эту проблему можно избежать, если автор не сможет вносить изменения пока редактор не совершит нужное количество прочтений.

Изолированность: чтение фантомов

|  |  |
| --- | --- |
| **Транзакция 1** | **Транзакция 2** |
|  | SELECT sum(attr\_2)  FROM table\_1; |
| INSERT INTO table\_1 (attr\_1, attr\_2)  VALUES (15, 20); |  |
| COMMIT; |  |
|  | SELECT sum(attr\_2)  FROM table\_1; |

При повторном чтении в рамках одной транзакции будут считаны строки, добавленные параллельно выполнявшейся транзакцией.

Пример: Редактор делает изменения в документе, созданном автором, но когда в основную копию документа производственным отделом вносятся изменения, может быть обнаружено, что автор добавил в документ новый, неотредактированный материал. Аналогично ситуации неповторяющегося чтения, эту проблему можно избежать, если никто не добавляет новый материал в документ, пока редактор и производственный отдел не закончат работать с оригиналом документа.

**Уровни изоляции транзакций**

* Влияют на блокировки, накладываемые при чтении данных
* Определяют, можно ли считывать данные, добавленные или измененные незафиксированной транзакцией
* Чем ниже уровень изоляции – тем выше вероятность одновременного доступа пользователей к данным, но одновременно выше вероятность эффектов параллелизма

Уровни:

* Read uncommitted (чтение незафиксированных данных)
* Read committed (чтение фиксированных данных)
* Repeatable read (повторяемость чтения)
* Serializable (упорядочиваемость)

**Read uncommitted**

* Первый уровень изоляции
* Гарантирует отсутствие потерянных обновлений
* Итоговое значение – результат выполнения каждой транзакции
* Возможно считывание незафиксированных изменений
* Данные блокируются на время внесения изменений
* На время чтения данных блокировка отсутствует

**Read committed**

* Второй уровень изоляции
* Используется в большей части СУБД
* Защита от «грязного» чтения
* В процессе выполнения одна из транзакций успешно завершается, тогда остальные работают с измененными данными
* Реализация в жизни на усмотрение разработчиков СУБД:
  + Блокирование читаемых и изменяемых данных
  + Сохранение нескольких версий параллельно изменяемых строк

**Read committed:** блокировка

* Блокирование читаемых и изменяемых данных
* *Пишущая* транзакция блокирует данные для *читающих* транзакций уровня read commited или выше
* Отсутствует «грязное» чтение
* Возможно неповторяющееся чтение

**Read committed:** версионность

* Сохранение нескольких версий параллельно изменяемых строк
* СУБД создает новую версию строки, с которой работает «изменяющие данные» транзакция
* Для «читающей» транзакции доступна последняя **зафиксированная** версия

**Repeatable read**

* Третий уровень изоляции
* Читающая транзакция игнорирует изменения в данных, которые были ей ранее прочитаны
* Никакая транзакция не может изменять данные, читаемые текущей транзакцией, пока чтение не завершено
* Спасает от эффекта неповторяющегося чтения

**Serializable**

* Четвертый (самый высокий) уровень изоляции
* Транзакции **полностью** изолированы друг от друга
* Параллельных транзакций как будто бы не существует вовсе
* Транзакции не подвержены эффекту «фантомного чтения»

SET TRANSACTION ISOLATION LEVEL

{ READ UNCOMMITTED | READ COMMITTED | REPEATABLE READ | SERIALIZABLE }

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Уровни изоляции** | **Загрязнение данных** | **Грязное чтение** | **Неповторяемое чтение** | **Фантомы** |
| READ UNCOMMITTED | НЕТ | ДА | ДА | ДА |
| READ COMMITTED | НЕТ | НЕТ | ДА | ДА |
| REPEATABLE READ | НЕТ | НЕТ | НЕТ | ДА |
| SERIALIZABLE | НЕТ | НЕТ | НЕТ | НЕТ |

Пример 1:

CREATE TABLE #T (keycol int primary key, col1 int)

-- вставим значения и откатим транзакцию, хотя никакой ошибки не было. Просто мы не захотели вносить изменения

BEGIN TRAN;

INSERT INTO #T(keycol, col1) VALUES(4, 101);

INSERT INTO #T(keycol, col1) VALUES(5, 201);

SELECT \*

FROM #T

ROLLBACK TRAN;

SELECT \*

FROM #T

DROP TABLE #T

Первый SELECT покажет, что было вставлено. Второй покажет, что транзакция действительно откатилась.

Пример 2:

Несмотря на то, что вставить значение 6 можно, откатится вся транзакция.

CREATE TABLE #T (keycol int primary key, col1 int)

BEGIN TRAN;

INSERT INTO #T(keycol, col1) VALUES(4, 101);

INSERT INTO #T(keycol, col1) VALUES(5, 201);

COMMIT TRAN;

BEGIN TRAN;

INSERT INTO #T(keycol, col1) VALUES(6, 101);

INSERT INTO #T(keycol, col1) VALUES(5, 201);

ROLLBACK TRAN;

SELECT \* FROM #T

DROP TABLE #T

Сравните, что будет, если Вы во второй транзакции ROLLBACK замените на COMMIT.

Пример транзакции:

CREATE TABLE #T (keycol int primary key, col1 int)

-- вставим значения

BEGIN TRAN;

INSERT INTO #T(keycol, col1) VALUES(4, 101);

INSERT INTO #T(keycol, col1) VALUES(5, 201);

COMMIT TRAN;

SELECT \* FROM #T

-- Менять значения id и val для тестирования вывода сообщений об ошибках.

DECLARE @id int, @val float

SET @id = 7 -- неповторный ключ для тестирования транзакции

--SET @id = 4 -- повторный ключ для того, чтобы вставка в транзакции сломалась

--SET @val = 2 -- для тестирования ошибки транзакции

SET @val = 0 -- чтобы в catch попадали по делению на ноль, если вставка данных прошла.

-- знакомый Вам из программирования блок TRY-CATCH.

BEGIN TRY

BEGIN TRAN;

SELECT 1 Где, @@TRANCOUNT Сколько -- проверка значения переменной

INSERT INTO #T(keycol, col1) VALUES(6, 201); -- новое значение ключа

INSERT INTO #T(keycol, col1) VALUES(@id, 201); -- тестовая вставка, в зависимости от id транзакция либо откатится, либо нет.

SELECT 2 Где, @@TRANCOUNT Сколько

COMMIT TRAN; -- если всё ок, то транзакция пройдет, если ошибка при вставке, то попадаем в catch сразу из ошибки.

SELECT 3 Где, @@TRANCOUNT Сколько

PRINT 10/@val -- если вставка прошла успешно, то попадание в catch происходит, если val = 0

END TRY

BEGIN CATCH

SELECT 4 Где, @@TRANCOUNT Сколько

-- Подумайте, зачем нужно это условие на служебную переменную @@Trancount?

IF @@TRANCOUNT > 0

ROLLBACK TRAN;

DECLARE @ErrorMessage NVARCHAR(4000);

DECLARE @ErrorSeverity INT;

DECLARE @ErrorState INT;

-- с 2012 версии для проброски текста ошибки существует THROW.

SELECT

@ErrorMessage = ERROR\_MESSAGE(),

@ErrorSeverity = ERROR\_SEVERITY(),

@ErrorState = ERROR\_STATE();

-- посмотрите, что будет, если закомментировать RAISEERROR: транзакция откатится, но сообщения об ошибке не будет.

RAISERROR (@ErrorMessage, -- Message text.

@ErrorSeverity, -- Severity.

@ErrorState -- State.

);

END CATCH

SELECT 5 Где, @@TRANCOUNT Сколько

SELECT \* FROM #T

DROP TABLE #T

@@TRANCOUNT – возвращает число инструкций BEGIN TRANSACTION, выполненных в текущем соединении

Инструкция BEGIN TRANSACTION увеличивает значение @@TRANCOUNT на 1.

Инструкция ROLLBACK TRANSACTION уменьшает значение @@TRANCOUNT до 0 (исключение – откат до savepoint)*.*

Инструкции COMMIT TRANSACTION и COMMIT WORK уменьшают значение @@TRANCOUNT на 1.

Переменная @@Trancount показывает уровень вложенности транзакции. 0 – в данный момент код не внутри транзакции. Более 1 – транзакция активна.

Функция xact\_state() показывает состояние транзакции. 0 – транзакция неактивная, 1 – есть незафиксированная транзакция, которая может быть зафиксирована, -1 – есть незафиксированная транзакция, но она не может быть зафиксирована из-за предшествующей ошибки.

Задание: посмотрите, что будет происходить, если перебирать различные варианты переменных @id и @val, что будет, если убрать условие на переменную @@TRANCOUNT, что будет, если закомментировать сообщение RAISERROR?

Попробуйте самостоятельно разобраться с THROW (в версии старше 2012).