**Побочные эффекты параллелизма.**

Все операции в базе происходят не мгновенно и при одновременном изменении данных различными пользователями возможны следующие побочные эффекты:

* Потерянное обновление (lost update)
* «Грязное» чтение (dirty read)
* Неповторяющееся чтение (non-repeatable read)
* Фантомное чтение (phantom reads)

Далее, эти эффекты рассматриваются подробно и приводятся SQL скрипты, показывающие проблему на практике. Шаги по подготовки и особенности запуска скриптов описаны ниже.

**Требования для запуска скриптов**

* Первым нужно запускать скрипт для транзакции №1, а затем сразу же скрипт для транзакции №2 (не позднее чем через 10 секунд после начала выполнения первого скрипта).
* В базе должна существовать таблица с именем Table1 и колонками Id и Value. В ней ожидается наличие одной строки:
* Для создания таблицы и наполнения её данными можно запустить следующий скрипт.

IF (EXISTS (SELECT \* FROM INFORMATION\_SCHEMA.TABLES WHERE TABLE\_SCHEMA = 'dbo' AND TABLE\_NAME = 'Table1'))

DROP TABLE Table1

CREATE TABLE Table1 (Id INT IDENTITY, Value INT)

INSERT INTO Table1 (Value) VALUES(1)

* Так же данный скрипт желательно выполнить перед рассмотрения каждого примера. Это будет гарантировать идентичность получаемых результатов с теми, что описаны ниже.

**Потерянное обновление (lost update)**

Эффект проявляется при одновременном изменении одного блока данных разными транзакциями. Причём одно из изменений может теряться.   
Данная формулировка может по-разному интерпретироваться.

**Потерянное обновление – Интерпретация №1**

Две транзакции выполняют одновременно UPDATE для одной и той же строки, и изменения, сделанные одной транзакцией, затираются другой.

|  |  |  |
| --- | --- | --- |
|  | Транзакция 1 | Транзакция 2 |
|  | UPDATE Table1  SET Value = Value + 5  WHERE Id = 1;    SELECT Value  FROM Table1  WHERE Id = 1; | UPDATE Table1  SET Value = Value + 7  WHERE Id = 1;    SELECT Value  FROM Table1  WHERE Id = 1; |
| Результат: | Value = 6 | Value = 8 |

*Почему так происходит?*

Прежде чем выполнить обновление, обе транзакции читают значение в колонке Value – оно равно 1. Предположим, что транзакция 2 успевает записать значение первой, тогда новое значение в колонке Value будет 8 (1+7). Затем транзакция 1 так же вычисляет новое значение, но для расчёта использует ранее вычитанное значение (1). В итоге после завершения транзакции 1 в колонке Value окажется 6 (1+5), а не 13 (1+7+5).

К счастью в MS SQL данный сценарий невозможен, потому что даже самый низкий уровень изоляции предотвращает такую ситуацию и результатом всегда будет 13, а не 8.

**Потерянное обновление – Интерпретация №2**

Сценарий аналогичен первому, но значение Value вычитывается во временную переменную

|  |  |  |
| --- | --- | --- |
|  | Транзакция 1 | Транзакция 2 |
|  | BEGIN TRAN;    DECLARE @Value INT;    SELECT @Value = Value  FROM Table1  WHERE Id = 1;    WAITFOR DELAY '00:00:10';    UPDATE Table1  SET Value = @Value + 5  WHERE Id = 1;    COMMIT TRAN;    SELECT Value  FROM Table1  WHERE Id = 1; | BEGIN TRAN;    DECLARE @Value INT;    SELECT @Value = Value  FROM Table1  WHERE Id = 1;        UPDATE Table1  SET Value = @Value + 7  WHERE Id = 1;    COMMIT TRAN;    SELECT Value  FROM Table1  WHERE Id = 1; |
| Результат: | Value = 6 | Value = 8 |

**«Грязное» чтение (dirty read)**

Это такое чтение, при котором могут быть считаны добавленные или изменённые данные из другой транзакции, которая впоследствии не подтвердится (откатится).   
Так как данный эффект возможен только при минимальном уровне изоляции, а по умолчанию используется более высокий уровень изоляции (READ COMMITTED), то в скрипте чтения данных уровень изоляции будет явно установлен как READ UNCOMMITTED. Если вернуть уровень изоляции по умолчанию (READ COMMITTED) для транзакции 2, то поведение поменяется.

|  |  |  |
| --- | --- | --- |
|  | Транзакция 1 | Транзакция 2 |
|  | BEGIN TRAN;    UPDATE Table1  SET Value = Value \* 10  WHERE Id = 1;    WAITFOR DELAY '00:00:10';    ROLLBACK;    SELECT Value  FROM Table1  WHERE Id = 1; | SET TRANSACTION ISOLATION LEVEL READ UNCOMMITTED;  --SET TRANSACTION ISOLATION LEVEL READ COMMITTED;    BEGIN TRAN;    SELECT Value  FROM Table1  WHERE Id = 1;    COMMIT TRAN; |
| Результат для READ UNCOMMITTED | Value = 1 | Value = 10 |
| Результат для READ COMMITTED | Value = 1 | Value = 1 |

Мы видим, что внутри второй транзакции было вычитано значение 10, которое никогда не было успешно сохранено в базу (оно было отклонено командой ROLLBACK).

**Неповторяющееся чтение (non-repeatable read)**

Проявляется, когда при повторном чтении в рамках одной транзакции, ранее прочитанные данные, оказываются изменёнными. Данный эффект может наблюдаться при уровне изоляции ниже, чем REPEATABLE READ.

|  |  |  |
| --- | --- | --- |
|  | Транзакция 1 | Транзакция 2 |
|  | SET TRANSACTION ISOLATION LEVEL READ COMMITTED  --SET TRANSACTION ISOLATION LEVEL REPEATABLE READ    BEGIN TRAN;    SELECT Value  FROM Table1  WHERE Id = 1;    WAITFOR DELAY '00:00:10';    SELECT Value  FROM Table1  WHERE Id = 1;    COMMIT; | BEGIN TRAN;    UPDATE Table1  SET Value = 42  WHERE Id = 1;    COMMIT TRAN; |
| Результат для READ COMMITTED | Value = 1  Value = 42 | Мгновенное выполнение |
| Результат для REPEATABLE READ | Value = 1  Value = 1 | Ожидание завершения транзакции 1 |

**Фантомное чтение (phantom reads)**

Можно наблюдать, когда одна транзакция в ходе своего выполнения несколько раз выбирает множество строк по одним и тем же критериям. При этом другая транзакция в интервалах между этими выборками добавляет или удаляет строки, или изменяет столбцы некоторых строк, используемых в критериях выборки первой транзакции, и успешно заканчивается. В результате получится, что одни и те же выборки в первой транзакции дают разные множества строк. Данный эффект можно наблюдать, когда уровень изоляции ниже чем SERIALIZABLE.

|  |  |  |
| --- | --- | --- |
|  | Транзакция 1 | Транзакция 2 |
|  | SET TRANSACTION ISOLATION LEVEL REPEATABLE READ  --SET TRANSACTION ISOLATION LEVEL SERIALIZABLE    BEGIN TRAN;    SELECT \* FROM Table1    WAITFOR DELAY '00:00:10'    SELECT \* FROM Table1    COMMIT; | BEGIN TRAN;    INSERT INTO Table1 (Value)  VALUES(100)    COMMIT TRAN; |
| Результат для REPEATABLE READ | *первый SELECT:*   |  |  | | --- | --- | | **Id** | **Value** | | 1 | 1 |   *второй SELECT:*   |  |  | | --- | --- | | **Id** | **Value** | | 1 | 1 | | 2 | 100 | | Мгновенное выполнение |
| Результат для SERIALIZABLE | *первый SELECT:*   |  |  | | --- | --- | | **Id** | **Value** | | 1 | 1 |   *второй SELECT:*   |  |  | | --- | --- | | **Id** | **Value** | | 1 | 1 | | Ожидание завершения транзакции 1 |

**Уровни изоляции**

Понимая смысл побочных эффектов, очень просто разобраться в назначении каждого уровня изоляции, т.к. они отличаются между собой количеством побочных эффектов.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Потерянное обновление | Грязное чтение | Неповторяющееся чтение | Фантомное чтение |
| Read uncommitted | Нет /Есть (\*) | Есть | Есть | Есть |
| Read committed  Read committed Snapshot (\*\*) | Нет /Есть (\*) | Нет | Есть | Есть |
| Repeatable read | Нет | Нет | Нет | Есть |
| Serializable  Snapshot (\*\*) | Нет | Нет | Нет | Нет |

(\*) – эффект присутствует только в случае, если он трактуется согласно описанию в разделе «Потерянное обновление – Интерпретация №2».

(\*\*) – для данных уровней изоляция достигается не при помощи блокировок, а при помощи создания копии изменяемых данных, которые на время транзакции помещаются в tempdb;

Когда-то Ли Кэмпбел однажды отлично сказал: «Вы должны понимать как минимум на один уровень абстракции ниже того уровня, на котором программируете». Именно поэтому, понимание реализации позволит максимально глубоко разобраться в теме и вы сможете правильно и эффективно пользоваться предлагаемым инструментом.