Транзакции и параллелизм

- Фиксация вносимых изменений
- SQL и многопользовательский режим работы

- Транзакция (transection) – это группа операторов SQL, выполняемых, как единое целое.
- Параллелизм (concurrency) относится к механизмам, с помощью которых СУБД предотвращает взаимное влияние операций, одновременно выполняемых над одними и теми же данными равными пользователями.

Фиксация вносимых изменений

- Среду базы данных легко представить в виде множества пользователей, постоянно вводящих и изменяющих информацию. При этом система будет функционировать без сбоя.
- В реалии постоянно возникают ошибки, причиной которых являет как человек, так и компьютер.
- В случае ошибки необходимо дать возможность отменить выполнение операции.

Фиксация вносимых изменений

- Оператор SQL, влияющий на обновления (например на оператор обновления или на DROP TABLE), необязательно является необратимым.
- После выполнения оператора (или группы операторов) принимается решение станут ли произведенные изменения постоянными или будут прогнозируемыми;
- С этой целью операторы объединяются в группы, называемые **транзакциями**.

- Транзакция – это последовательность операторов SQL, которая принимается или отменяется, как единое целое.
- Инициируя сеанс работы в SQL, начинается транзакция;
- Все вводимые операторы будут входить в одну и ту же транзакцию, если не вводится оператор Commit Work или Rollback Work.
- Commit – делает все изменения, выполненные в ходе транзакции постоянными.
- Rollback –отменяет их.
- После каждого оператора Commit и Rollback начинается транзакция.

- Чтобы сделать постоянными все изменения с момента входа или последнего изменения Commit и Rollback, используется следующий синтаксис:

```
COMMIT WORK;
```

- Синтаксис для отмены изменений выглядит так:

```
ROLLBACK WORK;
```

- Во многих реализациях предусмотрен специальный параметр AUTOCOMMIT или SET AUTOCOMMIT ON
- Что бы вернутся к обычной обработки транзакций нужно ввести:

SET AUTOCOMMIT OFF

Система может автоматически включать AUTOCOMMIT при входе.

- При аварийном завершении сеанса пользователя – например, когда система дает сбой или пользователь перезагружает компьютер – астматически выполняется откат текущей транзакции.
- Это одна из причин по которым стоит делить операторы, вводимые в ручную, на множество отдельных транзакций.
- Транзакция не должна содержать много несвязанных операторов, она часто состоит из единственного оператора.

- Транзакции, которые включают целую группу несвязанных операторов позволяют только сохранить или отменить всю группу.
- Как группу, тогда как отмена обычно нужна только для одного конкретного изменения.
- Транзакция должна состоять из одного или нескольких связанных операторов.

- Предположим, что нам нужно удалить из базы данных продавца по имени Иванов. Прежде чем удалить ее саму оператором DELETE из таблицы ПРОДАВЦЫ, необходимо что-то сделать с его заказами и покупателями.
- Логично установить ID Продавца для этих заказов в NULL, чтобы никто из продавцов не получал по ним комиссионные, а покупателей передать продавцу Петрову.
- После этого вы можете удалить его из таблицы Продавцы.

```
UPDATE Orders
    SET snum = NULL
    WHERE snum = 1004;
UPDATE Customers
    SET snum
    WHERE snum = 1004;
DELETE FROM Salespeople
    WHERE snum = 1004;
```

- Следовательно, приведенную группу операторов можно рассматривать как одну транзакцию.

- Можно предварить эту группу операторов COMMIT или завершить оператором COMMIT или ROLLBACK

SQL и многопользовательский режим работы

- Обычно SQL используется в многопользовательской среде, где в одно и то же время операции над БД могут выполнять несколько пользователей.
- Это создает предпосылки для конфликтов между разными операциями.

Предположим, что вы применяете к таблице продавцы следующий оператор:

```
UPDATE Salespeople
  SET comm = comm * 2
  WHERE sname LIKE ‘R%’:
```

Во время время его выполнения вводится следующий запрос:

```
SELECT city, AVG (comm)
   FROM Salespeople
   GROUP BY city;
```

- Будут ли отражать средние значения, полученные последним пользователем, те изменения, которые вносились в таблицу ранее?
- Это может не иметь большого значения, здесь важно чтобы результаты запроса отражали либо все изменения, либо никаких.
- Любой промежуточный результат будет случайным образом завесить от того порядка, в котором производятся физические изменения данных.
- Результаты запросов не должны завесить от физических деталей и не могут быть случайными и непредсказуемыми.

- Допустим, что вы нашли ошибку и выполняете откат изменений после того как последний пользователь получил свои результаты.
- Теперь его средние значения основаны, на изменениях, которые утратили силу, но он не может об это узнать.

Типовые проблемы параллелизма

Одновременная обработка транзакций называется **параллелизмом (concurrency)** и здесь могут возникать следующий проблемы.

- Обновления могут выполнятся независимо друг от друга.
- Например, продавец выполняет запрос к инвентарной таблице, находит на складе 10 единиц товара и заказывает для покупателя 6 из них.
- До того, как это изменение будет сделано, другой продавец просматривает таблицу и заказывает 7 того же товара для своего покупателя.

- Изменения в БД могут отменяться уже после их использования, как в приведенном выше примере, когда вы отменили ошибочный ввод после получения пользователем его результатов.
- Частичный результат одной операции может влиять на результат другой операции, в том же примере пользователь получил средние значения пока вы производили обновления.
- Это не всегда представляет проблему, но во многих случаях функции, на подобие агрегатных, должны отражать состояние базы данных в момент относительной стабильности.

Например:

- Аудитор должен иметь возможность вернуться назад и опередить, что средние значения существовали в некоторый момент и могла оставаться неизменными, если бы после этого не вносилось никаких изменений.
- Это невозможно если во время вычисления функции выполнялось обновление.
- Когда два пользователя пытаются выполнить мешающие друг другу действия, возможно тупиковая ситуация.

Пример:

Два пользователя одновременно пытаются изменить значения внешнего ключа и значение его родительского ключа.

Стандартные термины для проблем параллелизма

1. Потерянное обновление (LOST UPDATE)
2. Преждевременное чтение (DIRTY READ)
3. Неповторяющееся чтение (NON-REPETEABLE READ)
4. Фантомная вставка (PHANTOM INSERT)

Они объясняются в следующих таблицах

Потерянное обновление

Транзакция №1	Состояние базы данных	Транзакция №2
SELECT comm	comm = . 12	SELECT comm
FROM Salespeople		
WHERE snum = 1001;		
UPDATE Salespeople	comm = .10	
SET comm = . 10		
WHERE snum = 1001		
COMMIT WORK;		
	comm = .14	UPDATE Salespeople
		SET comm = .14
		WHERE snum = 1001;
		COMMIT WORK;

- По завершению этой последовательности операторов значение comm = . 14
- Обновление до . 10 не имело результата.
- Это не обязательно является ли это проблемой, например установка для comm значения . 14 на следующей неделе дала бы тот же самых эффект перекрытия.
- Однако при изменении данных часто учитывается их предыдущее значение.
- Пользователь, выполнявший Транзакцию №2 был введет в заблуждение: он думал, что текущие comm = . 12, тогда как они были = . 10
- Если бы он намеревался увеличить комиссионные на . 02, то результат был бы ошибочным.
- Если выполнить увеличение непосредственно, использую в операторе UPDATE предложение SET comm = comm + 0.02, то ошибки можно избежать, но это не всегда легко сделать при сложных вычислениях.

Преждевременное чтение

Транзакция №1	Состояние базы данных	Транзакция №2
SELECT comm	comm = . 12	
FROM Salespeople		
WHERE snum = 1001;		
UPDATE Salespeople	comm = .10	
SET comm = . 10		
WHERE snum = 1001		
		SELECT comm
		FROM Salespeople
		WHERE snum = 1001;
ROLLBACK WORK	comm = .12	

- Запрос к транзакции №2 выводит значение, которое уже исчезло из БД.

- Отмена транзакции №1 эквивалентна тому, что значение comm никогда не = . 10, но именно это значение было показано в транзакции №2.

Неповторяющиеся чтение

Транзакция №1	Состояние базы данных	Транзакция №2
	comm = . 12	SELECT comm
		FROM Salespeople
		WHERE snum = 1001;
UPDATE Salespeople	comm = .10	
SET comm = . 10		
WHERE snum = 1001		
	comm = .10	SELECT comm
		FROM Salespeople
		WHERE snum = 1001;

- В транзакции №2 были получены два разных ответа на один вопрос, данные действительно изменились, но иногда необходимо гарантировать, что данные останутся постоянными до завершения транзакции
- Это особенно важно для приложений, которые выполняют чтение данных из обротку и сохранение новых значений тем или иным образом связанных со старыми.
- В токам случае подобная ситуация представляет проблему.

Фантомная вставка

Транзакция №1	Состояние базы данных	Транзакция №2
		SELECT AVG (comm)
		FROM Salespeople;
INSERT INTO Salespeople	Добавленная строка для продавца Иванова	
VALUES (1020 ‘Иванов’		
‘Москва’, . 15);		
		SELECT AVG (comm)
		FROM Salespeople;

- Поскольку в середине транзакции №2 выполняется оператор ISERT из транзакции №1 результаты одинаковых запросов будут отличатся.
- Здесь показан особый случай неповторяющегося чтения.
- Если вклинивающемся операторам, является INSERT, а не UPDATE или DELETE, то новая строка не может быть одной из тех, что выведены предыдущим запросом.
- Эта новая фантомная строка не существовавшая раньше.
- На практике различие между фантомными вставками и другими случаями неповторяющегося чтения заключается в том, что фантомные вставки дают более ограниченный эффект

Решение проблем параллелизма

- SQL обеспечивает управление параллелизмом CUNCURRENCY CONTROL
- В первоначальном стандарте SQL говорилось, что одновременное выполнение операторов должно быть организованно так, что бы это было эквивалентно ситуации, когда не один из операторов не вводится о полного завершения предыдущего (включая COMMIT и ROLLBACK)
- Это правило является идеалом с точки зрения целостности данных, но на практике, часто требуется обеспечить более

- В любой момент времени лишь одна транзакция будет иметь возможность изменять данные (хотя их чтение возможно для нескольких транзакций)
- Механизмы, которые SQL использует для управления параллельными операциями, называются **блокировками (LOCKS)**

- Блокировки приостанавливают определенные операции над БД на то время, пока активны другие операции или транзакции.

Все блокировки делятся на две общие категории:

- Пессимистические блокировки (pessimistic locks)

Прекращают доступ к данным для одновременных транзакций

- Оптимистические блокировки (optimistic locks)

Отслеживают возникновение конфликтов и при необходимости выполняют откат транзакции

Оптимистическое блокирование более подходит, если ожидаемая частота конфликта не велика. При таком блокировании операции могут выполняться с меньшими задержками, поскольку доступ к данным не ограничен.

Но при возникновении их конфликта все результаты работы пропадают, поэтому при высокой частоте конфликтов будет трудно доводить транзакции до конца.

Использование пессимистического блокирования

- При пессимистическом блокировании некоторые типы одновременного доступа к данным запрещены.
- Первая операция, которая может привести к конфликту, блокирует некоторые или все данные, которые использует, последующие конфликтные операции либо отменяются, либо ставятся в очередь до того момента, когда станет возможно их повторное выполнение.
- Используется конфигурационный параметр NO WAIT

Уровни изоляции

- Для поддержки блокировок определены уровни изоляции (ISOLATION LEVELS), определяющие какие типы конфликтов допустимы.
- В таблице перечислены различные уровни изоляции и операции разрешенные на каждом уровне.

Уровень изоляции	Потерянное обновление	Преждевременное чтение	Неповторяющееся чтение	Фантомная вставка
READ UNCOMMITED	нет	да	да	да
READ COMMITED	нет	нет	да	да
REPEATABLE READ	нет	нет	нет	да
SERIALIZABLE	нет	нет	нет	нет

- Уровень SERIALIZABLE (последовательное выполнение), устанавливается по умолчанию и обеспечивает максимальную степень контроля.
- В последовательном режиме каждая транзакция выполняется изолированно, не влияя на выполнение других параллельных транзакций.
- Эта ситуация идеальна с точки зрения целостности данных, но не является лучшей с точки зрения производительности.

- Уровень REPEATABLE READ (повторяющееся чтение) допускает из всех возможных видов неповторяющегося чтения только фантомные вставки.
- Этот уровень изоляции полезен на транзакции, на которые не влияет возможная вставка данных.
- Если добавление в таблицу новой строки не меняет результатов запросов данной транзакции.
- REPEATABLE READ окажется предпочтительнее SERIALIZABLE, поскольку допускает оператор INSERT.

- Уровень READ COMMITED (чтение с фиксацией) допускает неоднократное выполнение одного и того же запроса с разными результатами, но только при условии, что результаты параллельных транзакций были зафиксированы.

- Уровень READ UNCOMMITED (чтение без фиксаций) допускает не однократное выполнение одного и того же запроса с разными результатами независимо от того, были ли результаты параллельных транзакций зафиксированы.

Разделяемы и исключительные блокировки

- Для обеспечения всех уровней изоляции используются блокировки двух логических типов: разделяемые и исключительные
- Разделяемы блокировки (SHARED LOCKS) или S-блокировки (S-LOCKS) могут одновременно устанавливаться многими пользователями. Это позволяет любому количеству пользователей иметь доступ к данным, но не изменять их.
- Например, вы можете использовать разделяемые блокировки для повторного чтения.

- Исключительные блокировки (EXCLUSIVE LOCKS) или X-блокировки (X-LOCKS), позволяют иметь доступ к данным только владельцу блокировки.
- Исключительные блокировки используются для операторов изменяющих содержимое или структуру таблицы и позволяют исключить потерянные изменения.
- Можно одновременно устанавливать данные блокировки обоих типов.

- СПАСИБО ЗА ВНИМАНИЕ!