СБЕР УНИВЕРСИТЕТ

Data Engineer

SQL

Транзакции. Блокировки.

Преподаватель: Свирин Андрей Викторович
svirin.a.vi@sberbank.ru

Занятие 6

Для чего транзакции нам нужны

- Базы данных системы должны обеспечить одновременный и независимый доступ к данным для сотен и тысяч пользователей.
- При параллельной работе пользователей результаты вычислений могут зависеть от временных соотношений между действиями пользователей.
- Как избежать этой зависимости?
- Можно ли организовать чтение данных одним пользователем и одновременно их изменение другим?
- Как добиться, чтобы деньги, снятые с одного счета, и из-за сбоя системы не зачисленные на другой счет или не могли пропасть бесследно?

Транзакция

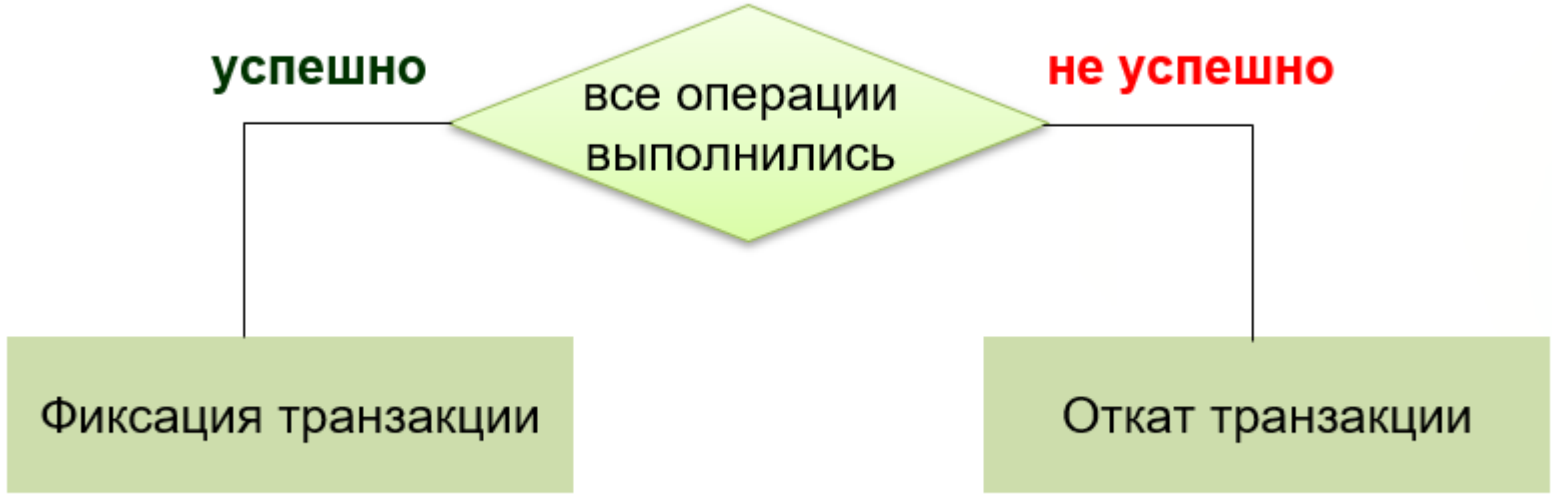
- Транзакция – это последовательность операций, проводимых над БД, выполняемых как единое целое и переводящих БД из одного непротиворечивого состояния в другое непротиворечивое состояние.
- Количество операций, входящих в транзакцию, может быть любым от одной до сотен, тысяч
- DE решает, какие команды должны выполняться как одна транзакция, а какие могут быть разбиты на несколько последовательно выполняемых транзакций.

Требование к транзакции - ACID

Требованиям к транзакциям, сокращенно называемые ACID

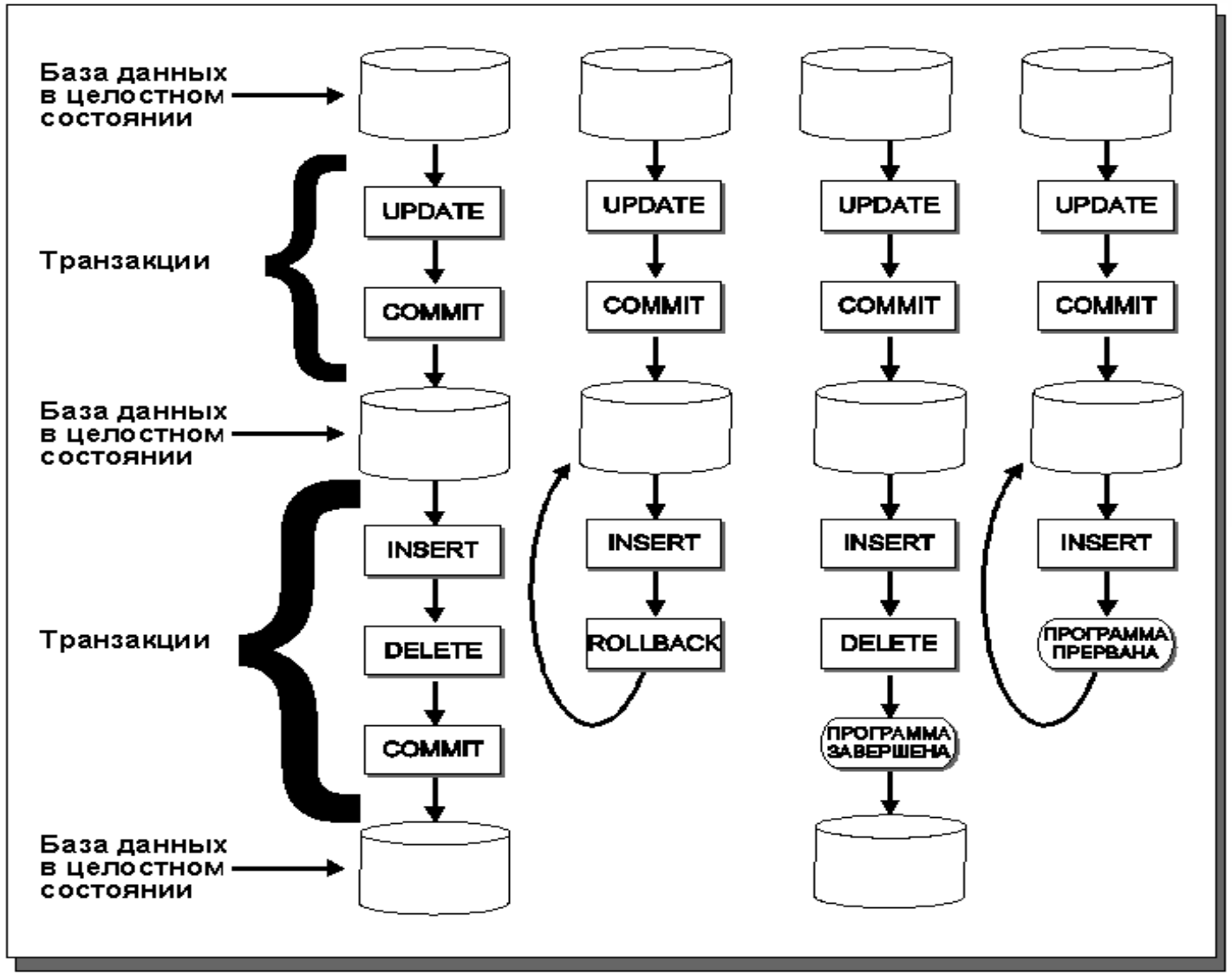
- Atomicity (Атомарност) - Транзакция не может выполниться частично, либо все, либо ничего
- Consistency (Согласованность) - После выполнения транзакции все данные должны находиться в согласованном состоянии
- Isolation (Изолированность) - Транзакция должна быт автономной и воздействовать на другие транзакции или завысить от них
- Durability (Устойчивость) - После завершения транзакции, внесенные изменения останутся неизменными

Транзакция может ..



- Фиксация транзакции – это действия, обеспечивающие сохранение на диске изменений БД, сделанные в процессе выполнения транзакции
- Откат транзакции – это действия, обеспечивающие аннулирование всех изменений БД, сделанные в процессе выполнения транзакции

Модели транзакции



Модели транзакции

Модель транзакций, отличная от предлагаемой ANSI/ISO, использует 4 оператора:

- ✓BEGIN TRANSACTION - сообщает о начале транзакции;
- ✓COMMIT TRANSACTION - сообщает об успешном завершении транзакции;
- ✓SAVE TRANSACTION - создает внутри транзакции точку сохранения (SAVEPOINT);
- ✓ROLLBACK - либо отменяет изменения после SAVEPOINT, возвращая транзакцию к месту, где был выполнен SAVE TRANSACTION, либо отменяет все изменения после BEGIN TRANCTION.

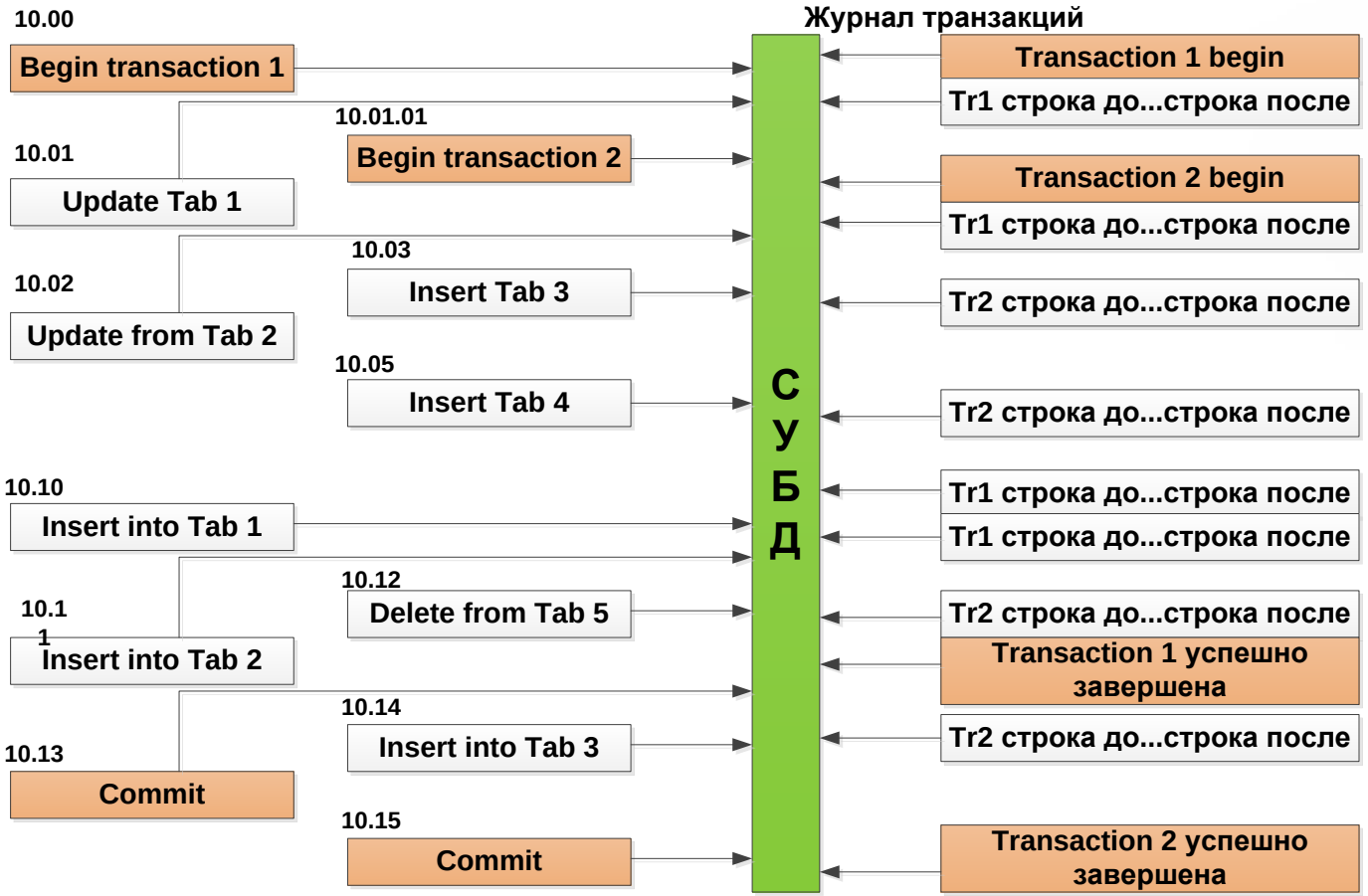
ANSI/ISO модель транзакций:

для начала транзакции не требуется выполнение специальных действий;

транзакция начинается автоматически вместе с первым оператором SQL или непосредственно после окончания предыдущей транзакции.

Журнал транзакции

Журнал транзакций – это специальным образом организованный последовательный файл, в котором фиксируются все изменения, выполняемые всеми транзакциями.



В случае системного сбоя с помощью специальной утилиты восстановления можно просмотреть журнал транзакций, отыскать незавершенные транзакции и даже отменить их.

3 Вида транзакции

Явная

Требуется явно указать команды начала и конца транзакций

Автоматическая

Каждая команда рассматривается как отдельная транзакция

Неявная

Транзакция начинается с первого оператора SQL и заканчивается явным указанием конца транзакции

СБЕР

3

Вида определение транзакции

Явная

Требуется явно указать команды начала и конца транзакций

Автоматическая

Каждая команда рассматривается как отдельная транзакция

Неявная

Транзакция начинается с первого оператора SQL и заканчивается явным указанием конца транзакции

Уровни изоляции транзакции

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

Уровни изоляции транзакции

Уровень изолированности	Фантомное чтение	Неповторяющееся чтение	«Грязное» чтение	Потерянное обновление
READ UNCOMMITTED	допустимо	допустимо	допустимо	Не будет
READ COMMITTED	допустимо	допустимо	Не будет	Не будет
REPEATABLE READ	допустимо	Не будет	Не будет	Не будет
SERIALIZABLE	Не будет	Не будет	Не будет	Не будет

SNAPSHOT отличается от Serialized в том, что не используются блокировки, но в результате фиксация изменений может оказаться невозможной, если параллельная транзакция изменила те же самые данные раньше; в этом случае вторая транзакция при попытке выполнить COMMIT вызовет сообщение об ошибке и будет отменена.

Фантомная чтение

транзакции №2 были получены два разных ответа на один и тот же запрос, так как данные действительно изменились, но иногда необходимо гарантировать, что данные останутся постоянными до завершения транзакции

Транзакция №1	Состояние базы данных	Транзакция №2
		SELECT AVG (comm)
		FROM Salespeople;
INSERT INTO Salespeople	Добавленная строка для продавца Свирин	
VALUES (11 ‘Свирин’		
‘Москва’, . 85);		
		SELECT AVG (comm)
		FROM Salespeople;

Неповторяющееся чтение

транзакции №2 были получены два разных ответа на один и тот же запрос, так как данные действительно изменились, но иногда необходимо гарантировать, что данные останутся постоянными до завершения транзакции

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;

«Грязное» чтение

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

Потерянное обновление

Транзакция №1 (желает уменьшить на 2)	Состояние базы данных	Транзакция №2 (желает увеличить на 2)
SELECT comm	comm = . 12	SELECT comm
FROM Salespeople		
WHERE snum = 1001;		
UPDATE Salespeople		
SET comm = . 10		
WHERE snum = 1001		
COMMIT;	comm = .10	
		UPDATE Salespeople
		SET comm = .14
		WHERE snum = 1001;
	comm = .14	COMMIT;
	А почему не 12?	

СБЕР

Одновременная обработка транзакций называется **параллелизмом (concurrency).**

Транзакции READ ONLY и READ WRIGHT

READ ONLY - если известно, что изменение производится **не будут**

READ WRIGHT - если известно, что изменение производится **будут**, но она может выполнять, как запросы, так и изменения данных

```
Oracle:
SET TRANSACTION [ READ ONLY | READ WRITE ]
[ ISOLATION LEVEL [ SERIALIZE | READ COMMITED ]
[ USE ROLLBACK SEGMENT ‘segment_name’ ]
[ NAME ‘transaction_name’ ];
```

MS SQL:
BEGIN { TRAN | TRANSACTION } [{ transaction_name | @tran_name_variable } [WITH MARK ['description']]] [;]

Блокировки: Разделяемые и исключительные

3 режима работы с данными

Механизмы, которые SQL использует для управления параллельными операциями, называются **блокировками (LOCKS - замок), основные:**

Цель – предотвратить конфликты между транзакциями, обеспечивая при этом максимальную степень параллелизма при доступе к базе данных.

с замком

Исключительные блокировки (**EXCLUSIVE LOCKS**) или X-блокировки (X-LOCKS), позволяют иметь доступ к данным только владельцу блокировки, т.е. при этом другие пользователи не могут изменить эту запись, но могут читать.

с замком

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.

Без учета замков

Если явно указать NO-LOCK, то записи можно почитать

Перед выполнение каких-либо операций с некоторым объектом, транзакция должна заблокировать этот объект.

Согласование блокировок

Если на записи блокировка	Другие блокировки
No lock	X-Lock
	S-Lock
	No-Lock
S-Lock	S-Lock
	No-Lock
X-Lock	No-Lock

Предупреждающая (преднамеренная) блокировка

IX-блокировка (Intent eXclusive locks):

IS-блокировка (Intent Shared lock).

SIX-блокировка (Shared Intent eXclusive locks): параллельные считывания (но не обновления), при этом сама транзакция может обновлять.

U-блокировки (Update lock): накладывается перед изменением данных и перед наложением X-lock.

(timestamp).

Согласование блокировок

	Транзакция T2 пытается наложить на таблицу блокировку:				
Транзакция T1 наложила на таблицу блокировку:	IS	S	IX	SIX	X
IS	да	да	да	да	нет
S	да	да	нет	нет	нет
IX	да	нет	да	нет	нет
SIX	да	нет	нет	нет	нет
X	нет	нет	нет	нет	Нет
U	да	да	нет	нет	нет

Грануляция блокировок (объем блокировки)

level locking:

- page
- table
- dbspace
- tablespace
- item (столбец или строка)

Это определяется *грануляцией* (granularity) блокировки, называемой также *уровнем блокирования* (locking level).

1. **RID – блокировка уровня строки**
2. **KEY– блокировка уровня индекса (группа строк)**
3. **PAG– блокировка уровня страницы**
4. **EXT– блокировка уровня группы страницы**
5. **TAB– блокировка уровня таблицы**
6. **DB – блокировка уровня базы данных**

Взаимная блокировка – ДедЛок (deadlock)

Процесс 1 блокирует ресурс А.
Процесс 2 блокирует ресурс Б.
Процесс 1 пытается получить доступ к ресурсу Б.
Процесс 2 пытается получить доступ к ресурсу А.

Время	Запрос	Транзакция
T1	Читает строку 1. Устанавливает разделяемую блокировку на строку 1.	
T2	Читает строку 2. Устанавливает разделяемую блокировку на строку 2.	
T3		Редактирует строку 3. Устанавливает исключительную блокировку на строку 3.
T4	Пытается прочитать строку 3. Транзакция приостанавливается в ожидании ресурса.	
T5		Пытается отредактировать строку 2. Возникает взаимная блокировка.

Взаимная блокировка – ДедЛок (deadlock)

В Oracle также возникают взаимные блокировки. Основной причиной их возникновения являются **неиндексированные** внешние ключи.

При изменении главной таблицы сервер Oracle **полностью** блокирует подчиненную таблицу в двух случаях:

- при изменении первичного ключа в главной таблице подчиненная таблица блокируется (при отсутствии индекса по внешнему ключу);
- при удалении строки в главной таблице подчиненная таблица также полностью блокируется (при отсутствии индекса по внешнему ключу).

Практика - REPEATABLE READ

```
create table accounts
(account_number number primary key,
account_balance number);
```

Строка	Номер счета	Баланс счета, $
1	123	500
2	234	250
3	345	400
4	456	100

Действия

Первый пользователь выполняет запрос:
select sum(account_balance) from accounts;

Второй пользователь выполняет транзакцию, переводящую 400 $ со счета 123 на счет 456 (в тот момент времени, когда первый пользователь прочитал строку 2, но еще не прочитал остальные).

Как будет получен очевидный результат результат – 1250?

REPEATABLE READ на основе **блокировок**

Время	Запрос	Транзакция
T1	Читает строку 1, sum = 500 $. На строке 1 устанавливается разделяемая блокировка.	
T2	Читает строку 2, sum = 750 $. На строке 2 устанавливается разделяемая блокировка	
T3		Пытается изменить строку 1, но эта попытка блокируется. Транзакция приостанавливается, пока не сможет установить исключительную блокировку.
T4	Читает строку 3	
T5	Читает строку 4, выдает результат суммирования	
T6	Фиксируется транзакция.	
T7		Изменяет строку 1, устанавливая на соответствующую строку исключительную блокировку. Теперь баланс в этой строке имеет значение 100 $.
T8		Изменяет строку 4 устанавливая на соответствующую строку исключительную блокировку. Теперь баланс в этой строке становится равным 500 $. Транзакция фиксируется.

REPEATABLE READ на основе **версионности**

Время	Запрос	Транзакция
T1	Читает строку 1, sum = 500 $. На строке 1 устанавливается разделяемая блокировка.	
T2	Читает строку 2, sum = 750 $. На строке 2 устанавливается разделяемая блокировка	
T3		Пытается изменить строку 1, но эта попытка блокируется. Транзакция приостанавливается, пока не сможет установить исключительную блокировку.
T4	Читает строку 3	
T5	Читает строку 4, выдает результат суммирования	
T6	Фиксируется транзакция.	
T7		Изменяет строку 1, устанавливая на соответствующую строку исключительную блокировку. Теперь баланс в этой строке имеет значение 100 $.
T8		Изменяет строку 4 устанавливая на соответствующую строку исключительную блокировку. Теперь баланс в этой строке становится равным 500 $. Транзакция фиксируется.

Практика - SERIALIZABLE

create table a (x int) ;

create table b (x int) ;

	Сеанс 1	Сеанс 2
T1	Alter session set isolation_level=serializable;	
T2		Alter session set isolation_level=serializable;
T3	insert into a select count(*) from b;	
T4		insert into b select count(*) from a;
T5	commit;	
T6		commit;

Практика - SERIALIZABLE

Результат – две таблицы с значениями 0 и 0.

```
alter session
set isolation_level=serializable
```

варианты?

Операторы для управления транзакциями

COMMIT
ROLLBACK SAVEPOINT <точка сохранения>
ROLLBACK TO <точка сохранения>

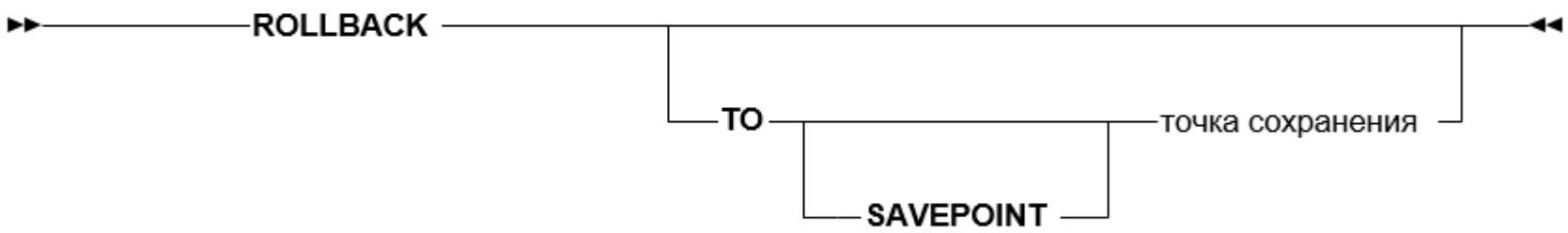
SET TRANSACTION

```
BEGIN
  COMMIT;

  SAVEPOINT POINT1;
    UPDATE EMP SET SAL = 3000 WHERE EMPNO = 7902;

  SAVEPOINT POINT2;
    SELECT SUM(SAL) INTO varSum FROM EMP;
    DBMS_OUTPUT.PUT_LINE('varSum1=' || varSum);

    UPDATE EMP SET SAL = SAL + 1000 WHERE EMPNO = 7788;

    SELECT SUM(SAL) INTO varSum FROM EMP;
    DBMS_OUTPUT.PUT_LINE('varSum2=' || varSum);


   IF varSum > 34000 THEN ROLLBACK TO POINT2; END IF;

  COMMIT;
   SELECT SUM(SAL) INTO varSum FROM EMP;
   DBMS_OUTPUT.PUT_LINE('varSum3=' || varSum);

END
```

Операторы для управления транзакциями

Команда фиксации транзакции

Команда отката транзакции



Команда установки точки сохранения

Автономная транзакция

Автономные транзакции позволяют создать новую транзакцию в пределах текущей,
так что можно фиксировать или откатывать ее изменения независимо от родительской транзакции.
Они позволяют приостановить текущую транзакцию, начать новую, выполнить ряд действий, зафиксировать их или откатить, не влияя на состояние текущей транзакции.

Используются для:
Для фиксации динамических ошибок.
Для аудита запрещенных попыток изменения данных.

варианты?

Автономная транзакция

```
create or replace procedure SberInfo
        ( inInfoMessage   in varchar2,  inSource in varchar2 )
  is
        PRAGMA AUTONOMOUS_TRANSACTION;
  begin
        insert into debug_log(id, LogTime, Message)
        values (seq_debug_log.nextval, sysdate, inInfoMessage, inSource);
        commit;
  exception
     when others then
           return;
  end LogInfo;
```

Дискретная транзакция

Дискретные транзакции (DISCRETE TRANSACTION) – это транзакции, для которых не генерируется информация отката (rollback segment).

Они работают в условиях некоторых ограничений, а именно:

- не должны модифицировать один блок более одного раза за одну транзакцию;
- модифицируют небольшое количество блоков базы данных;
- не модифицируют данные, участвующие в долго идущих запросах;
- не опрашивают новых значений данных, после того как обновили эти данные;
- не участвуют в распределенных транзакциях;
- не могут выполнять вставки и обновления сразу в обе таблицы, вовлеченные в ограничение ссылочной целостности;
- не модифицируют таблиц, содержащих значения типа LONG.

Дискретные транзакции могут использоваться одновременно со стандартными транзакциями.

Дискретная транзакция запускается с помощью процедуры BEGIN_DISCRETE_TRANSACTION. DISCRETE_TRANSACTIONS_ENABLED должен быть установлен в TRUE.

Дискретная транзакция

1.В течение дискретной транзакции все изменения, которые она вносит в любые данные, откладываются до момента завершения этой транзакции.

2.Информация повторения генерируется, но сохраняется в отдельной области памяти.

3.Когда такая транзакция выдает запрос commit, ее информация повторения переписывается в файл журнала транзакций (вместе с другими групповыми операциями commit), а изменения в блоке базы данных применяются непосредственно к блоку.

4.Модифицированный блок записывается в файл базы данных обычным порядком.

5.Управление возвращается в приложение после завершения операции commit.

СБЕР

Data Engineer

Ваши вопросы

Преподаватель: Свирин Андрей Викторович
svirin.a.vi@sberbank.ru