



САМАРСКИЙ УНИВЕРСИТЕТ  
SAMARA UNIVERSITY

# Базы данных

## Лекция 9 Транзакции

Агафонов Антон Александрович  
д.т.н., доцент кафедры ГИИБ

Самара



- Транзакции:
  - Требования к транзакциям;
  - Журнал транзакций;
  - Обеспечения изолированности и атомарности. MVCC;
  - Уровни изоляции транзакций;
  - Проблемы совместного доступа к данным.





## Определение

Транзакция - это выполняемая от имени определенного пользователя или процесса последовательность операторов манипулирования данными, переводящая базу данных из одного целостного состояния в другое целостное состояние.

Транзакция — это операция, состоящая из одного или нескольких запросов к базе данных.

Суть транзакций — обеспечить корректное выполнение всех запросов в рамках одной транзакции, а также обеспечить механизм изоляции транзакций друг от друга для решения проблемы совместного доступа к данным.

Любая транзакция либо выполняется полностью, либо полностью отменяется.





```
START TRANSACTION;
```

```
UPDATE accounts  
SET balance = balance - 100.00  
WHERE name = 'Alice';
```

```
UPDATE accounts  
SET balance = balance + 100.00  
WHERE name = 'Bob';
```

```
COMMIT;
```





Требования к транзакционной системе, обеспечивающие наиболее надежную и согласованную ее работу (**ACID**):

- **Atomicity** — Атомарность;
- **Consistency** — Согласованность;
- **Isolation** — Изолированность;
- **Durability** — Долговечность.





**Атомарность** гарантирует, что никакая транзакция не будет зафиксирована в системе частично.

Вне зависимости от количества вовлекаемых в транзакцию операторов, транзакция должна представлять собой единую и неделимую логическую единицу работы.

Транзакция выполняется как атомарная операция - либо выполняется вся транзакция целиком, либо она целиком не выполняется. Транзакция не может быть выполнена частично, если в процессе выполнения транзакции возникает какой-то сбой, то все выполненные до этого момента изменения данных должны быть отменены. Транзакция считается успешно выполненной, если в процессе исполнения не возникла какая-либо аварийная ситуация, и все проверки ограничений целостности дали положительный результат.





**Согласованность** (Consistency). Транзакция переводит базу данных из одного согласованного (целостного) состояния в другое согласованное (целостное) состояние. Внутри транзакции согласованность базы данных может нарушаться. Согласованность достигается вследствие атомарности, т.к. завершенная транзакция гарантирует согласованность данных.

Говоря о согласованности результатов, имеется в виду, что в результате выполнения транзакции данные будут соответствовать заложенной в базе данных логике и правилам поддержания целостности данных.





**Изолированность** (Isolation). Транзакции должны выполняться независимо одна от другой. Транзакции должны быть недоступны промежуточные результаты других параллельно исполняемых транзакций, а также её промежуточные результаты недоступны другим транзакциям.

Другими словами, изолированная транзакция А может получить доступ к объекту, обрабатываемому транзакцией В, только после завершения В, и наоборот. Таким образом при одновременной работе двух пользователей с одними и теми-же данными они не замечают друг друга, как если бы они работали с этими данными строго по очереди.

Изолированность — требование дорогое, поэтому в реальных БД существуют режимы (уровни изоляции), не полностью изолирующие транзакцию.







**Долговечность** (Durability). Если транзакция выполнена, то результаты ее работы должны сохраниться в базе данных и не могут быть утеряны или отменены ни при каких обстоятельствах. В случае успешного выполнения транзакции все изменения гарантировано фиксируются в постоянной памяти, даже если в следующий момент произойдет сбой системы. В случае неуспешного завершения транзакции по любой причине - гарантировано отсутствие каких-либо связанных с ней изменений во внешней памяти.

Т.е. независимо от проблем на нижних уровнях (к примеру, обесточивание системы или сбои в оборудовании) изменения, сделанные успешно завершённой транзакцией, должны остаться сохранёнными после возвращения системы в работу. Другими словами, если пользователь получил подтверждение от системы, что транзакция выполнена, он может быть уверен, что сделанные им изменения не будут отменены из-за какого-либо сбоя.





Журнализация изменений (write ahead logging, WAL) — функция СУБД, которая сохраняет информацию, необходимую для восстановления базы данных в предыдущее согласованное состояние в случае логических или физических отказов.

В простейшем случае журнализация изменений заключается в последовательной записи во внешнюю память всех изменений, выполняемых в базе данных. Записывается следующая информация:

- сведения о старте транзакции и ее идентификатор;
- перечень объектов БД, на которые повлияла транзакция;
- описание каждой из входящих в состав транзакции операций обновления данных (вставка, обновление, удаление);
- предыдущее состояние объекта и новое состояние объекта.
- сведения о завершении транзакции.

Формируемая таким образом информация называется **журнал изменений** базы данных. Журнал содержит отметки начала и завершения транзакции, и отметки принятия контрольной точки.





Общий алгоритм использования журнала транзакций:

- изменения пишутся в журнал транзакций (состояние до и после) и изменяются страницы БД в памяти;
- в журнал транзакций сбрасывается маркер успешного завершения транзакции;
- журнал транзакций сбрасывается на диск;
- данные страниц БД пишутся на диск;
- данные страниц БД сбрасываются на диск.





Дополнительные возможности использования журнала транзакций:

- Восстановление на момент времени (Point in time recovery) – использование резервной копии и журнала транзакций для восстановления системы после сбоя;
- Репликация – механизм синхронизации содержимого нескольких копий данных (в т.ч., содержимого базы данных).



MVCC (MultiVersion Concurrency Control — управление параллельным доступом посредством многоверсионности) — один из механизмов СУБД для обеспечения параллельного доступа к базам данных, заключающийся в предоставлении каждому пользователю так называемого «снимка» базы, обладающего тем свойством, что вносимые пользователем изменения невидимы другим пользователям до момента фиксации транзакции. Этот способ управления позволяет добиться того, что пишущие транзакции не блокируют читающих, и читающие транзакции не блокируют пишущих.

- Разные пользователи могут одновременно работать с одними и теми же данными;
- Каждый пользователь видит свой изолированный срез данных;
- Изменения, вносимые пользователем, никому не видны до завершения транзакции.



## Прямое обновление. Шаг 1: Таблица

id	name	notes
1	Alice	Отправляет сообщение Бобу
2	Bob	Принимает сообщение Алисы
3	Eve	Прослушивает диалог Алисы и Боба



## Прямое обновление . Шаг 2: Сканирование

id	name	notes
1	Alice	Отправляет сообщение Бобу
2	Bob	Принимает сообщение Алисы
3	Eve	Прослушивает диалог Алисы и Боба

read




**update**

id	name	notes
1	Alice	Отправляет сообщение Бобу
2	Bob	Принимает сообщение Алисы
3	Eve	Прослушивает диалог Алисы и Боба





	id	name	notes
	1	Alice	Отправляет сообщение Бобу
	2	Bob	<b>Расшифровывает сообщение Алисы</b>
	3	Eve	Прослушивает диалог Алисы и Боба



insert	id	name	notes
	1	Alice	Отправляет сообщение Бобу
	2	Bob	<b>Расшифровывает сообщение Алисы</b>
	3	Eve	Прослушивает диалог Алисы и Боба
	4	Dave	<b>Новый участник</b>



## Прямое обновление . Шаг 6: Удаление

	id	name	notes
	1	Alice	Отправляет сообщение Бобу
	2	Bob	<b>Расшифровывает сообщение Алисы</b>
	3	Eve	Прослушивает диалог Алисы и Боба
	4	Dave	<b>Новый участник</b>



id	name	notes
1	Alice	Отправляет сообщение Бобу
2	Bob	<b>Расшифровывает сообщение Алисы</b>
4	<b>Dave</b>	<b>Новый участник</b>



	id	name	notes	
	1	Alice	Отправляет сообщение Бобу	
update	2	Bob	<b>Расшифровывает сообщение Алисы</b>	
delete	3	Eve	Прослушивает диалог Алисы и Боба	
insert	4	Dave	<b>Новый участник</b>	read



xmin	xmax	id	name	notes
100	0	1	Alice	Отправляет сообщение Бобу
101	0	2	Bob	Принимает сообщение Алисы
102	0	3	Eve	Прослушивает диалог Алисы и Боба

**TXID: 103**



**update**

xmin	xmax	id	name	notes
100	0	1	Alice	Отправляет сообщение Бобу
101	0	2	Bob	Принимает сообщение Алисы
102	0	3	Eve	Прослушивает диалог Алисы и Боба

**TXID: 103**



- update

xmin	xmax	id	name	notes
100	0	1	Alice	Отправляет сообщение Бобу
101	<b>103</b>	2	<i>Bob</i>	<i>Принимает сообщение Алисы</i>
102	0	3	Eve	Прослушивает диалог Алисы и Боба

TXID: 103







	xmin	xmax	id	name	notes
	100	0	1	Alice	Отправляет сообщение Бобу
- update	101	<b>103</b>	2	Bob	Принимает сообщение Алисы
	102	0	3	Eve	Прослушивает диалог Алисы и Боба
+ update	<b>103</b>	<b>0</b>	<b>2</b>	<b>Bob</b>	<b>Расшифровывает сообщение Алисы</b>

TXID: 103



## MVCC. Шаг 5: Обновление завершено

xmin	xmax	id	name	notes
100	0	1	Alice	Отправляет сообщение Бобу
<b>101</b>	<b>103</b>	2	<i>Bob</i>	<i>Принимает сообщение Алисы</i>
102	0	3	Eve	Прослушивает диалог Алисы и Боба
<b>103</b>	<b>0</b>	<b>2</b>	<b>Bob</b>	<b>Расшифровывает сообщение Алисы</b>

TXID: 104



xmin	xmax	id	name	notes
100	0	1	Alice	Отправляет сообщение Бобу
<i>101</i>	<b>103</b>	2	<i>Bob</i>	<i>Принимает сообщение Алисы</i>
102	0	3	Eve	Прослушивает диалог Алисы и Боба
<b>103</b>	<b>0</b>	<b>2</b>	<b>Bob</b>	<b>Расшифровывает сообщение Алисы</b>
<b>104</b>	<b>0</b>	<b>4</b>	<b>Dave</b>	<b>Новый участник</b>

TXID: 104

insert





## MVCC. Шаг 7: Добавление завершено

xmin	xmax	id	name	notes
100	0	1	Alice	Отправляет сообщение Бобу
<b>101</b>	<b>103</b>	2	<i>Bob</i>	<i>Принимает сообщение Алисы</i>
102	0	3	Eve	Прослушивает диалог Алисы и Боба
<b>103</b>	<b>0</b>	<b>2</b>	<b>Bob</b>	<b>Расшифровывает сообщение Алисы</b>
<b>104</b>	<b>0</b>	<b>4</b>	<b>Dave</b>	<b>Новый участник</b>

TXID: 105



## MVCC. Шаг 8: Удаление

	xmin	xmax	id	name	notes	TXID: 105
	100	0	1	Alice	Отправляет сообщение Бобу	
	101	<b>103</b>	2	Bob	Принимает сообщение Алисы	
	102	105	3	Eve	Прослушивает диалог Алисы и Боба	
	103	0	2	Bob	Расшифровывает сообщение Алисы	
	104	0	4	Dave	Новый участник	



## MVCC. Шаг 9: Удаление завершено

xmin	xmax	id	name	notes
100	0	1	Alice	Отправляет сообщение Бобу
<i>101</i>	<b>103</b>	2	<i>Bob</i>	<i>Принимает сообщение Алисы</i>
<i>102</i>	<i>105</i>	3	<i>Eve</i>	<i>Прослушивает диалог Алисы и Боба</i>
<b>103</b>	<b>0</b>	<b>2</b>	<b>Bob</b>	<b>Расшифровывает сообщение Алисы</b>
<b>104</b>	<b>0</b>	<b>4</b>	<b>Dave</b>	<b>Новый участник</b>

TXID: 106



Пример блокировки в PostgreSQL. Подтверждение первой транзакции

```
BEGIN TRANSACTION
ISOLATION LEVEL REPEATABLE READ;

SELECT balance
FROM accounts WHERE name = 'Alice';
```

	balance
	100

```
UPDATE accounts
SET balance = balance + 100
WHERE name = 'Alice';
```

*UPDATE 1*

```
COMMIT;
```

```
BEGIN TRANSACTION
ISOLATION LEVEL REPEATABLE READ;

SELECT balance
FROM accounts WHERE name = 'Alice';
```

	balance
	100

```
UPDATE accounts
SET balance = balance + 100
WHERE name = 'Alice';
```

*Waiting...*

*ERROR: ОШИБКА: не удалось сериализовать  
доступ из-за параллельного изменения*





Пример блокировки в PostgreSQL. Откат первой транзакции

```
BEGIN TRANSACTION
ISOLATION LEVEL REPEATABLE READ;

SELECT balance
FROM accounts WHERE name = 'Alice';
```

	balance
	200

```
UPDATE accounts
SET balance = balance + 100
WHERE name = 'Alice';
```

*UPDATE 1*

```
ROLLBACK;
```

```
BEGIN TRANSACTION
ISOLATION LEVEL REPEATABLE READ;

SELECT balance
FROM accounts WHERE name = 'Alice';
```

	balance
	200

```
UPDATE accounts
SET balance = balance + 100
WHERE name = 'Alice';
```

*Waiting...*

```
UPDATE 1
COMMIT;
```







Пример блокировки в MySQL. Подтверждение первой транзакции

```
SET TRANSACTION ISOLATION LEVEL  
    REPEATABLE READ;
```

```
START TRANSACTION;
```

```
SELECT balance  
FROM accounts WHERE name = 'Alice';
```

	balance
	100

```
UPDATE accounts  
SET balance = balance + 100  
WHERE name = 'Alice';
```

*UPDATE 1*

```
COMMIT;
```

```
SET TRANSACTION ISOLATION LEVEL  
    REPEATABLE READ;
```

```
START TRANSACTION;
```

```
SELECT balance  
FROM accounts WHERE name = 'Alice';
```

	balance
	100

```
UPDATE accounts  
SET balance = balance + 100  
WHERE name = 'Alice';
```

*Waiting...*

```
UPDATE 1  
COMMIT;
```





При параллельном выполнении транзакций возможны следующие проблемы:

- **потерянное обновление** (lost update) — при одновременном изменении одного блока данных разными транзакциями теряются все изменения, кроме последнего;
- **«грязное» чтение** (dirty read) — чтение незафиксированных изменений, осуществленных другой транзакцией. В случае перезаписи этих промежуточных значений или отката первой транзакции незафиксированные изменения могут быть отменены, а прочитавшая их транзакция с этого момента станет работать с неверными данными;
- **неповторяющееся чтение** (non-repeatable read) — при повторном чтении в рамках одной транзакции ранее прочитанные данные оказываются изменёнными;
- **фантомное чтение** (phantom reads) — одна транзакция в ходе своего выполнения несколько раз выбирает множество строк по одним и тем же критериям. Другая транзакция в интервалах между этими выборками добавляет 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.





Под «уровнем изоляции транзакций» понимается степень обеспечиваемой внутренними механизмами СУБД защиты от всех или некоторых видов несогласованности данных, возникающих при параллельном выполнении транзакций.

Стандарт SQL-92 определяет шкалу из четырёх уровней изоляции.

- **READ UNCOMMITTED** (чтение незафиксированных данных) – наименее защищенный уровень изоляции, при котором транзакции способны читать незафиксированные изменения, сделанные другими транзакциями;
- **READ COMMITTED** (чтение фиксированных данных) – исключается «грязное» чтение, транзакция увидит только изменения, зафиксированные другими транзакциями;
- **REPEATABLE READ** (повторяющееся чтение) – накладывает блокировки на обрабатываемые транзакцией строки и не допускает их изменения другими транзакциями. В результате транзакция видит только те строки, которые были зафиксированы на момент ее запуска. Основной недостаток повторяемого чтения – высокая вероятность появления строк-фантомов;
- **SERIALIZABLE** (сериализуемость) – самый надежный уровень изоляции, полностью исключающий взаимное влияние транзакций.





## Уровни изоляции транзакций

Уровень изоляции	Потерянное обновление	«Грязное» чтение	Неповторяющееся чтение	Фантомное чтение	Аномалии сериализации
Read uncommitted	Исключено	Возможно	Возможно	Возможно	Возможно
Read committed	Исключено	Исключено	Возможно	Возможно	Возможно
Repeatable read	Исключено	Исключено	Исключено	Возможно	Возможно
Serializable	Исключено	Исключено	Исключено	Исключено	Исключено



## Потерянное обновление (Lost Update)

Одна транзакция переписывает изменения, осуществленные другой транзакцией, в результате одно из изменений будет утеряно.

Транзакция 1	Транзакция 2
<pre>UPDATE accounts SET balance = balance + 100 WHERE name = 'Alice';</pre>	<pre>UPDATE accounts SET balance = balance + 500 WHERE name = 'Alice';</pre>



## «Грязное» чтение (Dirty Read)

Чтение (незафиксированных) данных, добавленных или изменённых еще не завершённой транзакцией.

Транзакция 1	Транзакция 2
<pre>SELECT balance FROM accounts WHERE name = 'Alice'; ----- 100</pre>	
<pre>UPDATE accounts SET balance = balance + 100 WHERE name = 'Alice';</pre>	
	<pre>SELECT balance FROM accounts WHERE name = 'Alice'; ----- 200</pre>
<pre>ROLLBACK;</pre>	





## Неповторяющееся чтение (Non-Repeatable Read)

При повторном чтении в рамках одной транзакции ранее прочитанные данные оказываются изменёнными.

Транзакция 1	Транзакция 2
	<pre>SELECT balance FROM accounts WHERE name = 'Alice'; ----- 100</pre>
<pre>UPDATE accounts SET balance = balance + 100 WHERE name = 'Alice';</pre>	
<pre>COMMIT;</pre>	
	<pre>SELECT balance FROM accounts WHERE name = 'Alice'; ----- 200</pre>





## Фантомное чтение (Phantom Reads)

Ситуация, когда при повторном чтении в рамках одной транзакции одна и та же выборка дает разные множества строк.

От неповторяющегося чтения оно отличается тем, что результат повторного обращения к данным изменился не из-за изменения/удаления самих этих данных, а из-за появления новых (фантомных) данных.

Транзакция 1	Транзакция 2
	<code>SELECT SUM(balance) FROM accounts;</code> ----- 500
<code>INSERT INTO accounts (name, balance)</code> <code>VALUES ('Dave', 1000);</code>	
<code>COMMIT;</code>	
	<code>SELECT SUM(balance) FROM accounts;</code> ----- 1500







Ситуация, когда параллельное выполнение транзакций приводит к результату, невозможному при последовательном выполнении тех же транзакций.

Транзакция 1	Транзакция 2
<pre>SELECT SUM(balance) FROM accounts WHERE name = 'Bob'; ----- 1000  INSERT INTO accounts (name, balance) VALUES ('Alice', 1000);  COMMIT;</pre>	<pre>SELECT SUM(balance) FROM accounts WHERE name = 'Alice'; ----- 500  INSERT INTO accounts (name, balance) VALUES ('Bob', 500);  COMMIT;</pre>





Уровень изоляции	Потерянное обновление	«Грязное» чтение	Неповторяющееся чтение	Фантомное чтение	Аномалии сериализации
Read uncommitted	Исключено	Возможно	Возможно	Возможно	Возможно
Read committed	Исключено	Исключено	Возможно	Возможно	Возможно
Repeatable read	Исключено	Исключено	Исключено	Возможно	Возможно
Serializable	Исключено	Исключено	Исключено	Исключено	Исключено

### Важно

Более высокий уровень изоляции транзакций уменьшает количество аномалий за счет увеличения количества блокировок и вероятности отката транзакции.





По умолчанию, как только пользователь отправляет СУБД инструкцию SQL, сервер самостоятельно стартует транзакцию, осуществляет фиксацию изменений, если в ходе выполнения транзакции не произошло исключительных ситуаций, или осуществляет автоматический откат операций, осуществленных в транзакции, при наличии в них хотя бы одной ошибки. Такое поведение сервера называют **неявным управлением транзакциями**.

В SQL также предусматривается инструментальный набор по **явному управлению транзакциями**:

- 1) `START TRANSACTION` (или `BEGIN`) – используется для запуска новой транзакции;
- 2) `COMMIT` – фиксация изменений, осуществленных транзакцией;
- 3) `ROLLBACK` – откат транзакции в исходное состояние.



## START TRANSACTION

```
COMMIT [AND [NO] CHAIN] [[NO] RELEASE]  
ROLLBACK [AND [NO] CHAIN] [[NO] RELEASE]
```

```
SET autocommit = {0 | 1}
```

Выражение **AND CHAIN** указывает системе, что с очередной транзакцией следует работать так, как будто она является следующим звеном предыдущей. В результате вторая транзакция станет использовать общие настройки первой транзакции (например, уровень изоляции).

Ключевое слово **RELEASE** требует, чтобы после завершения транзакции клиент завершил текущую сессию и отключился от сервера. По умолчанию действует режим **NO RELEASE** – сессия не прерывается.

Для некоторых операторов нельзя выполнить откат с помощью **ROLLBACK**. Это операторы языка определения данных (DDL). Сюда входят запросы **CREATE**, **ALTER**, **DROP**, **TRUNCATE**, **COMMENT**, **RENAME**.





**SAVEPOINT** identifier

**ROLLBACK TO [SAVEPOINT]** identifier

**RELEASE SAVEPOINT** identifier

Транзакция может быть разделена на точки сохранения.

Оператор **SAVEPOINT** устанавливает именованную точку сохранения транзакции с именем идентификатора. Если текущая транзакция имеет точку сохранения с тем же именем, старая точка сохранения удаляется, а новая устанавливается.

Оператор **ROLLBACK TO SAVEPOINT** откатывает транзакцию до указанной точки сохранения без прерывания транзакции. Изменения, которые выполняются в текущей транзакции для строк после установки точки сохранения, отменяются при откате.

Для удаления одной или нескольких точек сохранения используется команда **RELEASE SAVEPOINT**.



Явное задание уровня изоляции транзакции:

**SET [GLOBAL | SESSION] TRANSACTION ISOLATION LEVEL**

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



## Пример выполнения транзакции в MySQL

```
SET TRANSACTION ISOLATION LEVEL  
    READ COMMITTED;  
START TRANSACTION;
```

```
UPDATE accounts  
SET balance = balance + 100  
WHERE name = 'Alice';  
-----  
UPDATE 1  
COMMIT;
```

```
SET TRANSACTION ISOLATION LEVEL  
    READ COMMITTED;  
START TRANSACTION;  
  
SELECT balance  
FROM accounts WHERE name = 'Alice';  
-----  
100
```

```
SELECT balance  
FROM accounts WHERE name = 'Alice';  
-----  
200  
  
COMMIT;
```





## Пример выполнения транзакции в MySQL

```
SET TRANSACTION ISOLATION LEVEL  
    REPEATABLE READ;  
START TRANSACTION;
```

```
UPDATE accounts  
SET balance = balance + 100  
WHERE name = 'Alice';  
-----  
UPDATE 1  
COMMIT;
```

```
SET TRANSACTION ISOLATION LEVEL  
    REPEATABLE READ;  
START TRANSACTION;  
  
SELECT balance  
FROM accounts WHERE name = 'Alice';  
-----  
200
```

```
SELECT balance  
FROM accounts WHERE name = 'Alice';  
-----  
200  
  
COMMIT;
```







**САМАРСКИЙ** УНИВЕРСИТЕТ  
SAMARA UNIVERSITY

**БЛАГОДАРЮ  
ЗА ВНИМАНИЕ**

Агафонов А.А.  
д.т.н., доцент кафедры ГИИБ