



Базы данных

Лекция 4

Liquibase. Транзакции



Андрей Каледин

В предыдущих сериях...

В предыдущих сериях...

- Создание таблиц в БД
- Добавление, изменение, получение данных
- Индексы

План лекции

1. Технологии для изменения схемы данных в реальном мире
2. Зачем нужны транзакции
3. ACID
4. Уровни изоляции транзакций
5. Аномалии транзакций

Liquibase

Как менять схему данных в проде?

Как менять схему данных в проде?

Подключаться к БД через консоль или DBeaver и выполнять запросы руками

Как менять схему данных в проде?

Подключаться к БД через консоль или DBeaver и выполнять запросы руками

Плюсы:

- Просто и понятно

Как менять схему данных в проде?

Подключаться к БД через консоль или DBeaver и выполнять запросы руками

Плюсы:

- Просто и понятно

Минусы:

- Можно забыть это сделать → Потенциальный сбой на проде
- При откате релиза надо не забыть сделать обратные изменения
- В dev, qa и prod контурах надо делать одно и то же
- Историю запуска скриптов надо где-то отдельно хранить
- ...

Как менять схему данных в проде?

Подключаться к БД через консоль или DBeaver и выполнять запросы руками

Плюсы:

- Просто и понятно

Минусы:

- Можно забыть это сделать → Потенциальный сбой на проде
- При откате релиза надо не забыть сделать обратные изменения
- В dev, qa и prod контурах надо делать одно и то же
- Историю запуска скриптов надо где-то отдельно хранить
- ??? ТОЧНО ХОТИТЕ РУКАМИ ЛЕЗТЬ В ПРОДОВУЮ БД ???

Liquibase

Liquibase - инструмент для управления изменениями схемы базы данных

Liquibase

Liquibase - инструмент для управления изменениями схемы базы данных

Возможности:

1. Версионный контроль изменений

Liquibase

Liquibase - инструмент для управления изменениями схемы базы данных

Возможности:

1. Версионный контроль изменений
2. Контроль за многопоточными изменениями

Liquibase

Liquibase - инструмент для управления изменениями схемы базы данных

Возможности:

1. Версионный контроль изменений
2. Контроль за многопоточными изменениями
3. Возможность отката изменений

Liquibase

Liquibase - инструмент для управления изменениями схемы базы данных

Возможности:

1. Версионный контроль изменений
2. Контроль за многопоточными изменениями
3. Возможность отката изменений
4. Поддержка популярных СУБД: PostgreSQL, MongoDB, Clickhouse и т.д.

Liquibase

Liquibase - инструмент для управления изменениями схемы базы данных

Возможности:

1. Версионный контроль изменений
2. Контроль за многопоточными изменениями
3. Возможность отката изменений
4. Поддержка популярных СУБД: PostgreSQL, MongoDB, Clickhouse и т.д.
5. Логирование изменений (таблицы DatabaseChangeLog и DatabaseChangeLogLock)

Liquibase: пример

Содержимое файла changelog-master.xml

```
<databaseChangeLog ...>  
  <include file="changelog/create-tables.xml" relativeToChangelogFile="true"/>  
  <include file="changelog/add-new-column.sql" relativeToChangelogFile="true"/>  
</databaseChangeLog>
```

Содержимое файла create-tables.xml

Содержимое файла add-new-column.sql

Liquibase: пример

Содержимое файла changelog-master.xml

```
<databaseChangeLog ...>
  <include file="changelog/create-tables.xml" relativeToChangelogFile="true"/>
  <include file="changelog/add-new-column.sql" relativeToChangelogFile="true"/>
</databaseChangeLog>
```

Содержимое файла create-tables.xml

```
<databaseChangeLog ...>

  <changeSet id="create_users" author="alex">
    <createTable tableName="users">
      <column name="id" type="INT" autoIncrement="true">
        <constraints primaryKey="true" nullable="false"/>
      </column>
      <column name="username" type="VARCHAR(50)"/>
    </createTable>
  </changeSet>
```

Содержимое файла add-new-column.sql

```
</databaseChangeLog>
```

Liquibase: пример

Содержимое файла changelog-master.xml

```
<databaseChangeLog ...>
  <include file="changelog/create-tables.xml" relativeToChangelogFile="true"/>
  <include file="changelog/add-new-column.sql" relativeToChangelogFile="true"/>
</databaseChangeLog>
```

Содержимое файла create-tables.xml

```
<databaseChangeLog ...>

  <changeSet id="create_users" author="alex">
    <createTable tableName="users">
      <column name="id" type="INT" autoIncrement="true">
        <constraints primaryKey="true" nullable="false"/>
      </column>
      <column name="username" type="VARCHAR(50)"/>
    </createTable>
  </changeSet>

  <changeSet id="add_email_column" author="alex">
    <addColumn tableName="users">
      <column name="email" type="VARCHAR(100)"/>
    </addColumn>
  </changeSet>
</databaseChangeLog>
```

Содержимое файла add-new-column.sql

Liquibase: пример

Содержимое файла changelog-master.xml

```
<databaseChangeLog ...>
  <include file="changelog/create-tables.xml" relativeToChangelogFile="true"/>
  <include file="changelog/add-new-column.sql" relativeToChangelogFile="true"/>
</databaseChangeLog>
```

Содержимое файла create-tables.xml

```
<databaseChangeLog ...>

  <changeSet id="create_users" author="alex">
    <createTable tableName="users">
      <column name="id" type="INT" autoIncrement="true">
        <constraints primaryKey="true" nullable="false"/>
      </column>
      <column name="username" type="VARCHAR(50)"/>
    </createTable>
  </changeSet>

  <changeSet id="add_email_column" author="alex">
    <addColumn tableName="users">
      <column name="email" type="VARCHAR(100)"/>
    </addColumn>
  </changeSet>
</databaseChangeLog>
```

Содержимое файла add-new-column.sql

```
-- liquibase formatted sql
ALTER TABLE users
ADD COLUMN info TEXT;
-- rollback ALTER TABLE users DROP COLUMN info;
```

Альтернативы Liquibase

Это решение не единственное. В зависимости от стэка технологий и привычек команды еще используют:

- Flyway
- Django Migrations
- Alembic
- И т.д.



Did you do the migration ?



Yes...



What did it cost?

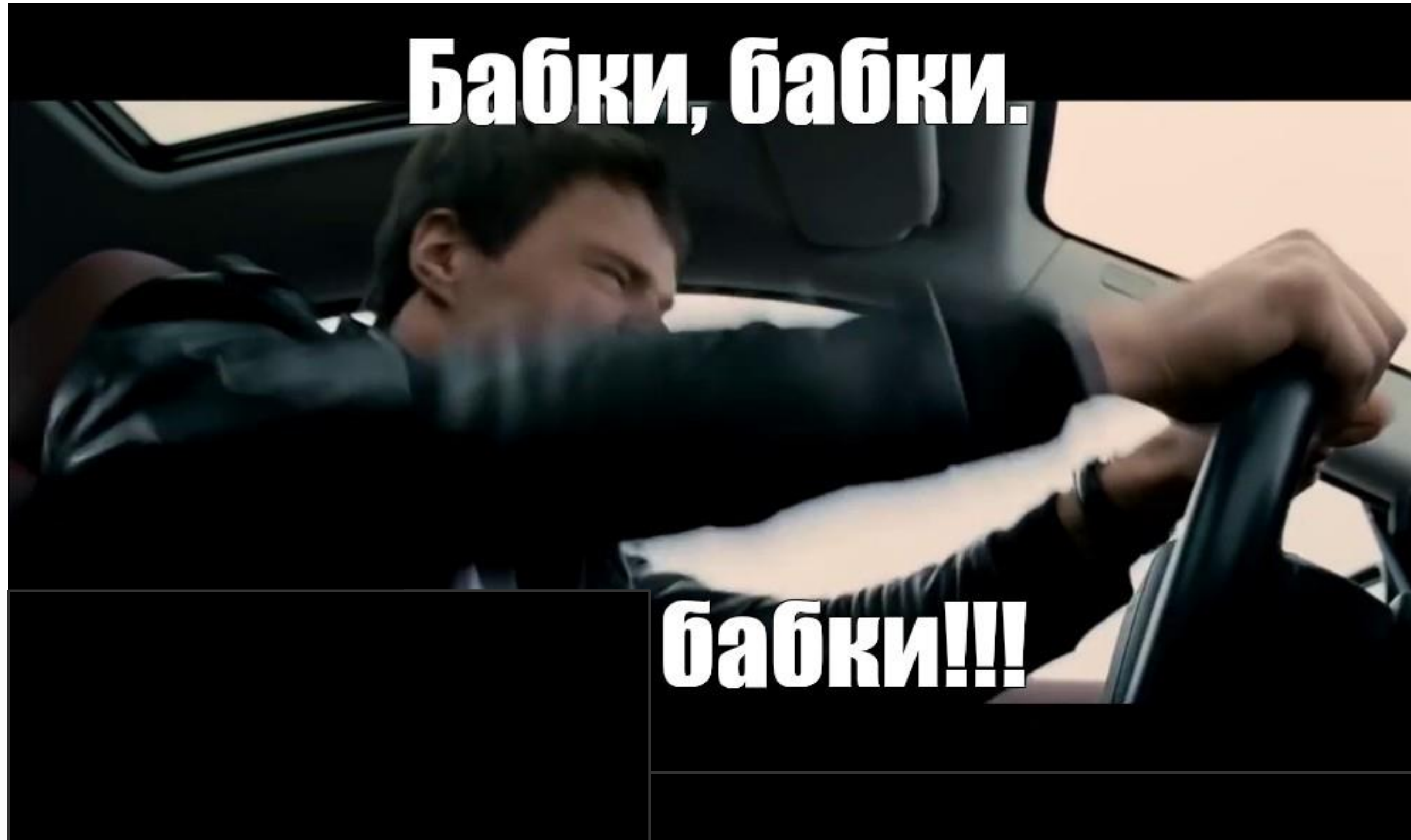


3 hours of downtime

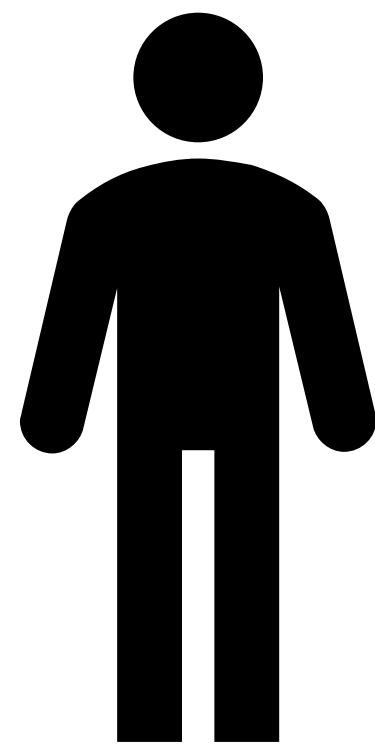
Транзакции

Зачем нужны транзакции в БД?

Зачем нужны транзакции в БД?



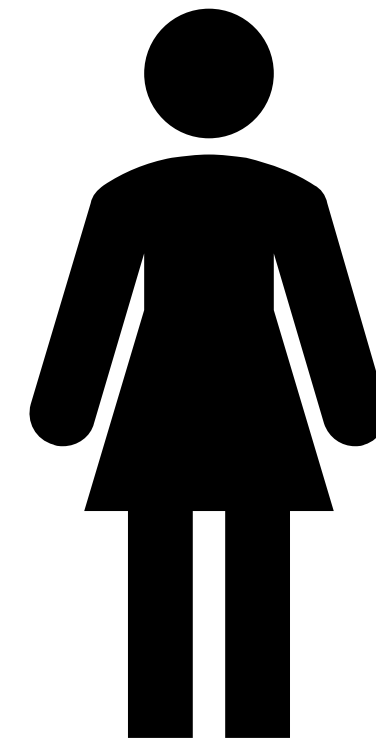
Зачем нужны транзакции в БД?



Паша



переводит деньги



Лена

Зачем нужны транзакции в БД?



Зачем нужны транзакции в БД?



$\text{money} = \text{money} - 10$

$\text{money} = \text{money} + 10$

Алгоритм перевода:

1. Спиши деньги у Паши
2. Начисли деньги Лене

Зачем нужны транзакции в БД?



Алгоритм перевода:

1. Спиши деньги у Паши
- 2. УПС, свет отключили**
- ~~3. Начисли деньги Лене~~

Зачем нужны транзакции в БД?



$\text{money} = \text{money} - 10$

$\text{money} = \text{money} + 10$

Алгоритм перевода:

1. Спиши деньги у Паши и начисли деньги Лене

ACID

ACID – набор правил, обеспечивающих надежность и целостность транзакций

ACID

ACID – набор правил, обеспечивающих надежность и целостность транзакций

Расшифровка:

- Atomicity (Атомарность)

Транзакция либо выполнилась целиком, либо отменилась целиком

ACID

ACID – набор правил, обеспечивающих надежность и целостность транзакций

Расшифровка:

- Atomicity (Атомарность)
Транзакция либо выполнилась целиком, либо отменилась целиком
- Consistency (Согласованность)
Транзакция не может привести БД из корректного состояния в некорректное

ACID

ACID – набор правил, обеспечивающих надежность и целостность транзакций

Расшифровка:

- Atomicity (Атомарность)
Транзакция либо выполнилась целиком, либо отменилась целиком
- Consistency (Согласованность)
Транзакция не может привести БД из корректного состояния в некорректное
- Isolation (Изолированность)
Транзакции не мешают друг другу

ACID

ACID – набор правил, обеспечивающих надежность и целостность транзакций

Расшифровка:

- Atomicity (Атомарность)
Транзакция либо выполнилась целиком, либо отменилась целиком
- Consistency (Согласованность)
Транзакция не может привести БД из корректного состояния в некорректное
- Isolation (Изолированность)
Транзакции не мешают друг другу
- Durability (Долговечность)
После выполнения транзакции, изменения точно сохраняются навсегда

ACID

Паша переводит деньги Лене:

- Atomicity (Атомарность)
Либо спишем у Паши и начислим Лене, либо не сделаем ничего
- Consistency (Согласованность)
Проверим, что у Паши точно достаточно денег ($\text{money} \geq 0$)
- Isolation (Изолированность)
Несколько параллельных переводов Паши не помешают друг другу
- Durability (Долговечность)
Выполненный перевод запомним навсегда

ACID

Что обещает ACID:

- Данные не потеряются
- Ограничения схемы данных не будут нарушены
- Многопоточная обработка работает корректно

Транзакция

Транзакция в БД – группа операций, выполняемых как единое целое. ACID – это свойства транзакций.

Девиз транзакций: «Все или ничего»

BEGIN;

UPDATE accounts
SET balance = balance - 100
WHERE username = 'Паша';

UPDATE accounts
SET balance = balance + 100
WHERE username = 'Лена';

COMMIT;

Как обеспечить ACID?

Как обеспечить ACID?

- «При старте одной из транзакций заблокировать любое использование другими транзакциями вовлеченных таблиц»
Сработает?

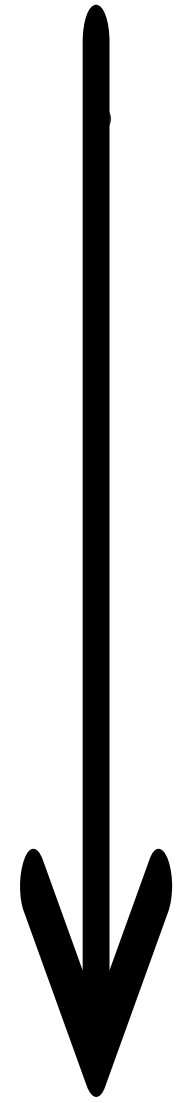
Как обеспечить ACID?

- «При старте одной из транзакций заблокировать любое использование другими транзакциями вовлеченных таблиц»
Сработает?
- Если у вас 1 запрос в час – сработает. А если 100 в секунду?

Как обеспечить ACID?

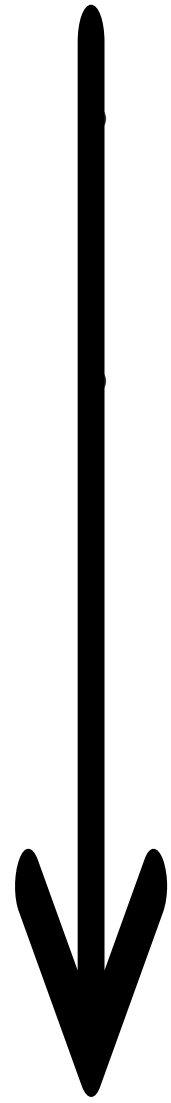
- «При старте одной из транзакций заблокировать любое использование другими транзакциями вовлеченных таблиц»
Сработает?
- Если у вас 1 запрос в час – сработает. А если 100 в секунду?
- Блокировать целую таблицу – это очень дорого. Нужно что-то подешевле. Но изолировать транзакции друг от друга все еще нужно

Уровни изоляции транзакций



READ UNCOMMITTED – транзакция видит незакоммиченные изменения другой транзакции

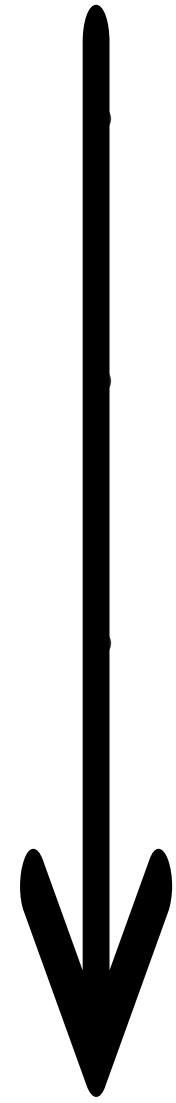
Уровни изоляции транзакций



READ UNCOMMITTED – транзакция видит незакоммиченные изменения другой транзакции

READ COMMITTED – транзакция читает только закоммиченные изменения других транзакций

Уровни изоляции транзакций

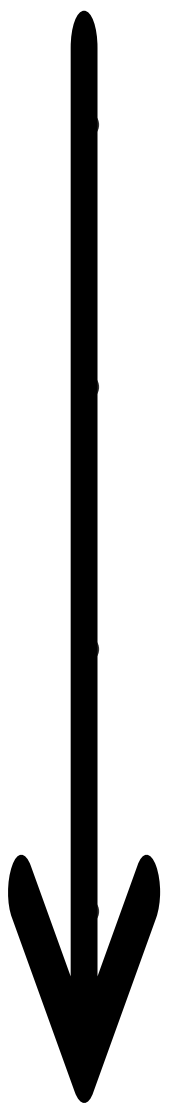


READ UNCOMMITTED – транзакция видит незакоммиченные изменения другой транзакции

READ COMMITTED – транзакция читает только закоммиченные изменения других транзакций

REPEATABLE READ – другие транзакции не меняют данные, которые прочитаны данной транзакцией

Уровни изоляции транзакций



READ UNCOMMITTED – транзакция видит незакоммиченные изменения другой транзакции

READ COMMITTED – транзакция читает только закоммиченные изменения других транзакций

REPEATABLE READ – другие транзакции не меняют данные, которые прочитаны данной транзакцией

SERIALIZABLE – блокировка любых действий в других транзакциях с данными этой транзакции

Аномалии транзакций

	Lost Update	Dirty Read	Non-Repeatable Read	Phantom Read	Serialization anomaly
Read Uncommitted	Невозможно 	Возможно 	Возможно 	Возможно 	Возможно 
Read Committed	Невозможно 	Невозможно 	Возможно 	Возможно 	Возможно 
Repeatable Read	Невозможно 	Невозможно 	Невозможно 	Возможно 	Возможно 
Serializable	Невозможно 	Невозможно 	Невозможно 	Невозможно 	Невозможно 

Lost Update (Потерянное обновление)

Транзакция 2 перетерла изменения Транзакции 1 до ее завершения

Транзакция 1	Транзакция 2
UPDATE X	
	UPDATE Y
COMMIT	
	COMMIT

Остались в итоге только изменения Y

Dirty Read (Грязное чтение)

Прочитали данные другой еще не завершенной транзакции

Транзакция 1	Транзакция 2
SELECT X -> 100	
UPDATE SET X = 200	
	SELECT X -> 200
ROLLBACK	
	COMMIT

Non-Repeatable Read (Неповторяющееся чтение)

При повторном чтении той же самой инфы получили другой результат

Транзакция 1	Транзакция 2
	SELECT X -> 100
UPDATE SET X = 200	
COMMIT	
	SELECT X -> 200
	COMMIT

Phantom Read (Чтение «фантомов»)

Одна и та же выборка дает разные значения. Отличается от предыдущей аномалии тем, что здесь аномалия возникла из-за добавления новых записей, а не изменения существующих

Транзакция 1	Транзакция 2
	SELECT COUNT(*) -> 10
INSERT	
COMMIT	
	SELECT COUNT(*) -> 11
	COMMIT

Serialization anomaly (Аномалия сериализации)

Параллельное выполнение транзакций приводит к результату, невозможному при их последовательном выполнении

Транзакция 1	Транзакция 2
SELECT SUM(VALUE) WHERE CLASS = 1 -> 30	SELECT SUM(VALUE) WHERE CLASS = 2 -> 10
INSERT (VALUE, CLASS) -> (30, 2)	INSERT (VALUE, CLASS) -> (10, 1)
COMMIT	COMMIT

Уровни изоляции транзакций в PostgreSQL

PostgreSQL поддерживает 3 уровня изоляции транзакций:

1. Read Committed
2. Repeatable Read
3. Serializable

TWO USERS BOOKED THE SAME ROOM



AT THE SAME TIME

Что мы сегодня узнали?

Что мы сегодня узнали?

1. Узнали, как менять схему данных в реальных системах
2. Познакомились с ACID
3. Рассмотрели разные уровни изоляции транзакций
4. Посмотрели на аномалии, которые могут возникать при многопоточной работе

Что будет на следующей лекции?

1. Партиции
2. Хранимые процедуры
3. Триггеры

Спасибо за внимание!



Анонимный опрос про курс:
<https://forms.gle/VGx1t3TGRyVh9Y3A8>



Беседа курса в Telegram