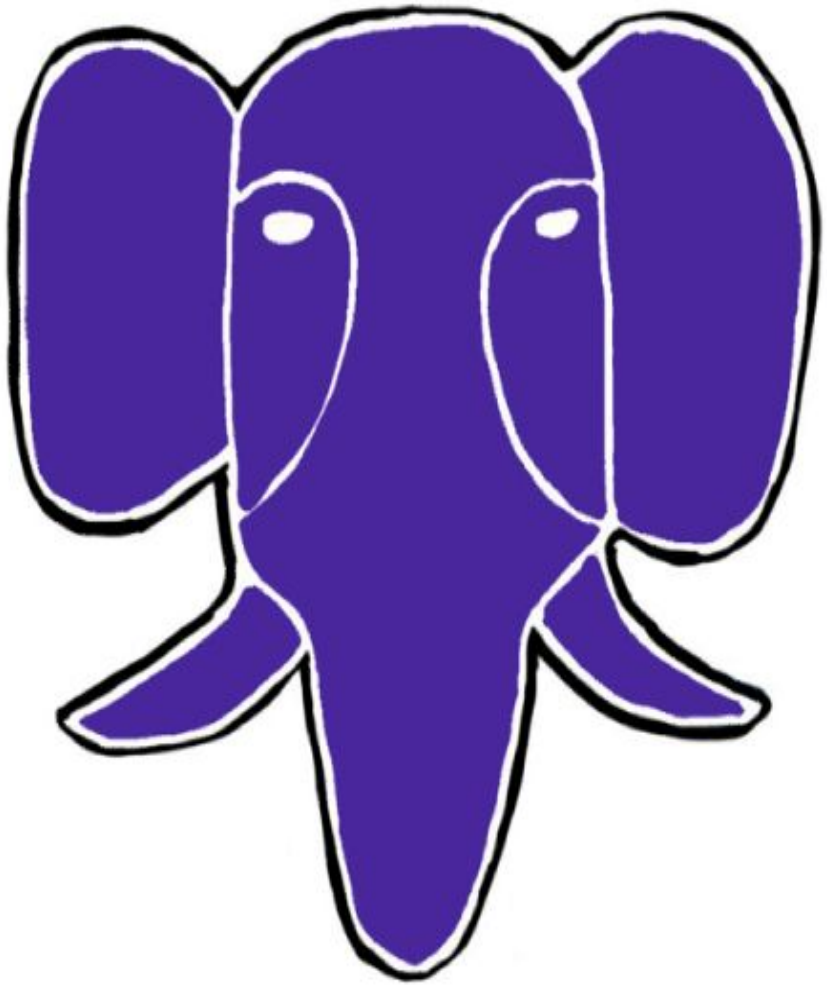
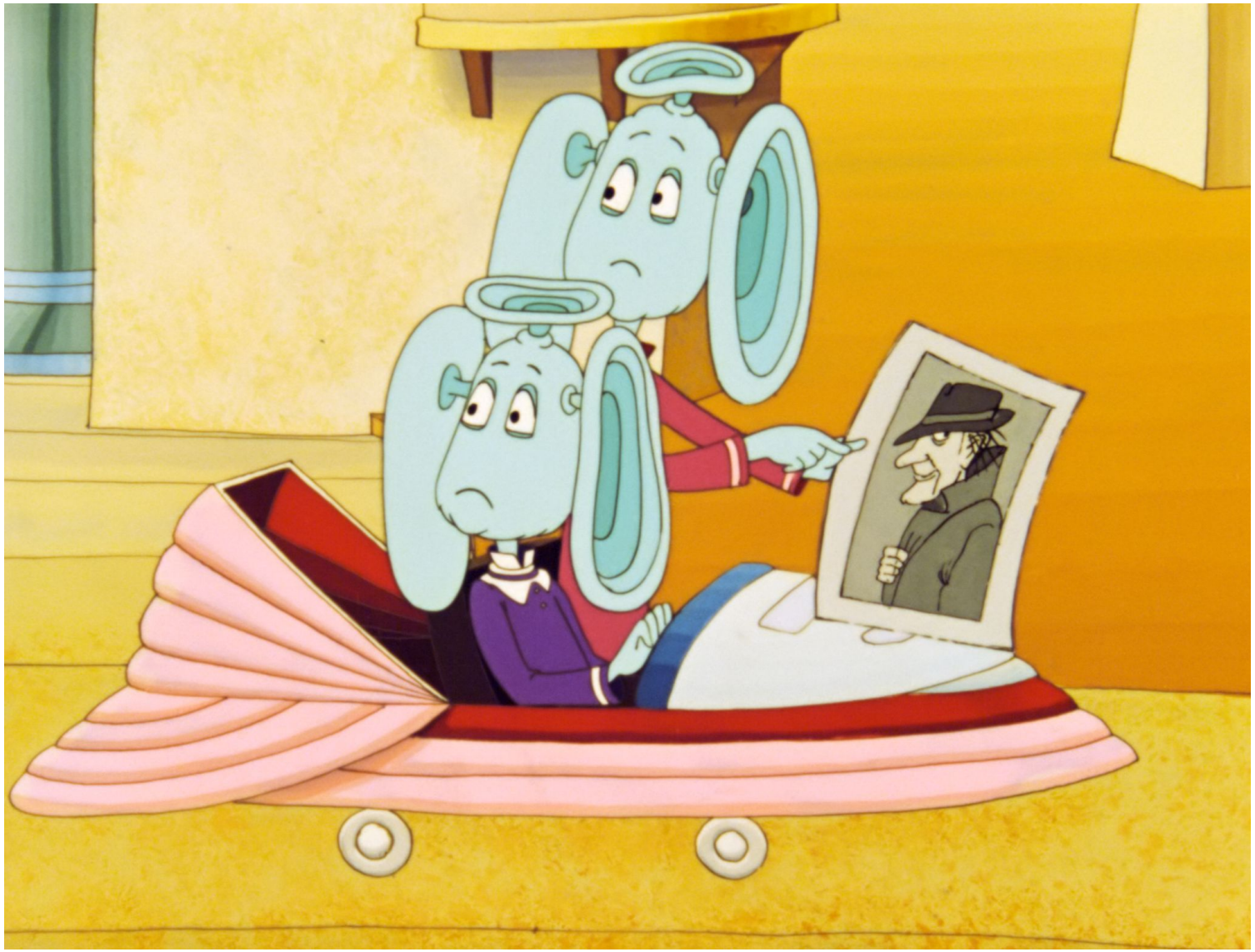
Уровни изоляции транзакций

Евгений Аристов



https://aristov.tech



Правила вебинара

Задаем вопрос в чат

Вопросы вижу, отвечу в момент логической паузы

Если есть вопрос голосом - поставьте знак ? в чат

Если остались вопросы, можно их задать на следующем занятии

Маршрут вебинара

- Транзакции
- Свойства транзакций ACID
- Уровни изоляции транзакций

https://aristov.tech

ACID

https://aristov.tech

Параллельная нагрузка

Для работы множества пользователей нам необходимо обеспечить:

- **параллельную** работу множества сессий
- **модификацию** данных, но так, чтобы пользователи **не мешали** друг другу ни с точки зрения **чтения** ни с точки зрения **записи**
- **целостность** данных

ACID

ACID

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

Обеспечивается это все с помощью транзакцией (**transaction**) - это одна или более операций, выполняемое приложением, со следующими свойствами:

- транзакция выполнена полностью (атомарность)
- параллельные транзакции не мешают друг другу (изолированность)
- транзакция переводит базу данных из одного корректного состояния в другое корректное состояние (согласованность)

OLTP - **O**n**l**ine **T**ransaction **P**rocessing - Транзакционные системы

https://aristov.tech

Транзакция

BEGIN; – начало транзакции

BEGIN TRANSACTION;

START TRANSACTION;

1 и более команд – при ошибке в одном из SQL, дальнейшие команды выполнены НЕ будут и PostgreSQL будет ждать команды ROLLBACK;

COMMIT; – подтверждение транзакции

COMMIT TRANSACTION;

ROLLBACK; – отмена транзакции

ROLLBACK TRANSACTION;

Уровни изоляции транзакций

Стандарт SQL допускает 4 уровня изоляции, которые определяются в терминах аномалий, которые допускаются при конкурентном выполнении транзакций на этом уровне:

- **dirty read** («грязное» чтение) Транзакция 1 может читать строки измененные, но еще не зафиксированные, транзакцией 2. **ROLLBACK** в транзакции 2 приведет к тому, что транзакция 1 *прочитает данные, которых никогда не существовало.*
- **non-repeatable read** (неповторяющееся чтение). После того, как транзакция 1 прочитала строку, транзакция 2 изменила или удалила эту строку и выполнила **COMMIT**. *При повторном чтении этой же строки транзакция 1 видит, что строка изменена или удалена.*
- **phantom read** (фантомное чтение). Транзакция 1 прочитала набор строк по некоторому условию. Затем транзакция 2 добавила строки, также удовлетворяющие этому условию. *Если транзакция 1 повторит запрос, она получит другую выборку строк*.
- **serialization anomaly** (аномалия сериализации). СУБД пытается выстроить транзакции последовательно во всех возможных комбинациях. При невозможности одного из вариантов происходит данная ошибка.

https://www.postgresql.org/docs/current/transaction-iso.html

Уровни изоляции транзакций

Аномалия сериализации. Классический пример из документации:

https://www.postgresql.org/docs/current/transaction-iso.html#XACT-SERIALIZABLE

class | value

-------+-------

1 | 10

1 | 20

2 | 100

2 | 200

Suppose that serializable transaction A computes:

SELECT SUM(value) FROM mytab WHERE class = 1;

and then inserts the result (30) as the `value` in a new row with `class = 2`. Concurrently, serializable transaction B computes:

SELECT SUM(value) FROM mytab WHERE class = 2;

and obtains the result 300, which it inserts in a new row with `class = 1`. Then both transactions try to commit.

Уровни изоляции транзакций

If either transaction were running at the Repeatable Read isolation level, both would be allowed to commit; but since there is no serial order of execution consistent with the result, using Serializable transactions will allow one transaction to commit and will roll the other back with this message:

ERROR: could not serialize access due to read/write dependencies among transactions

This is because if A had executed before B, B would have computed the sum 330, not 300, and similarly the other order would have resulted in a different sum computed by A.

Уровни изоляции транзакций

	dirty read	non-repeatable read	phantom read	serialization anomaly
Read Uncommitted	-	+	+	+
Read Committed	-	+	+	+
Repeatable Read	-	-	-	+
Serializable	-	-	-	-

На всех уровнях не допускается потеря зафиксированных изменений, то есть реализовывается буква D - Durability, но это если выставлен synchronous_commit в уровень local или выше

https://aristov.tech/blog/urovni-izolyaczii-tranzakczij/

Практика

https://aristov.tech

Итоги

https://aristov.tech

Итоги

Остались ли вопросы?

Спасибо!

Спасибо за внимание!

Когда дальше и куда?
материалы для бесплатного доступа будут появляться на ютубе
курс по Оптимизации PostgreSQL
продвинутое индивидуальное обучение менторинг

Аристов Евгений