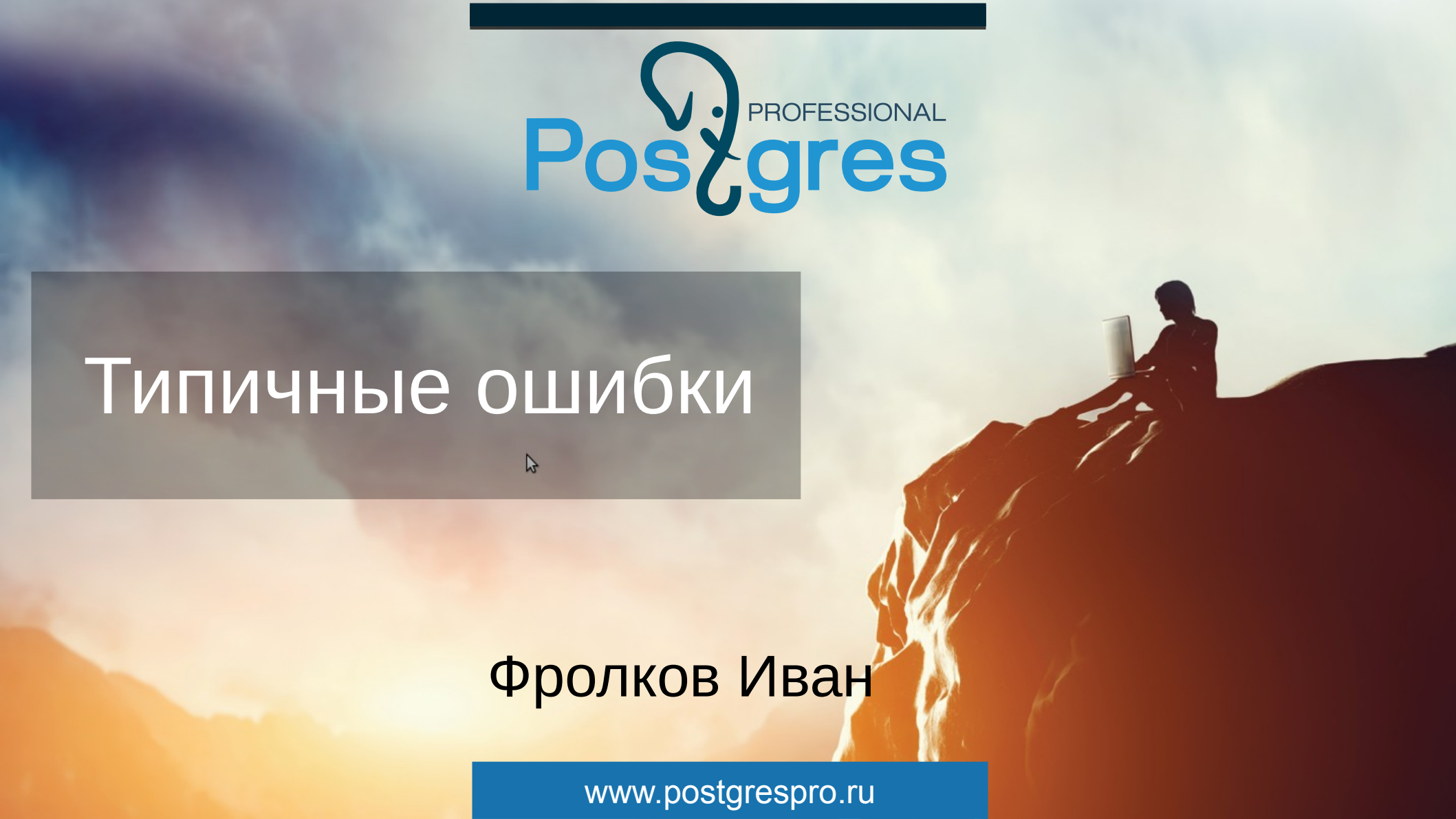
PROFESSIONAL Postgres

Типичные ошибки



Фролков Иван

www.postgrespro.ru

Postgres PROFESSIONAL

О чем пойдет речь

- Это не только про Postgres
  - Хотя и про него, разумеется, тоже
- Некоторые вопросы дискуссионны
  - А некоторые нет
- Веселиться, оказывается, можно как угодно, главное — настойчивость
- Вообще настойчивость — вещь обязательная; но не надо себе придумывать лишнюю работу, у нас ее и так предостаточно
- Я не самый умный

Непоследовательное именование

- Будьте последовательны
  - Это поможет как другим, так и вам самим
- id_user, user_id, userid, user, id, «idUser»
- Или вот еще
  - Таблицы department и fizlico в одной базе. Ну и так далее...

Выводы

- Именуйте объекты единообразно
- В идеальном случае название объекта очевидно без обращения к документации
- В плохом случае требует написания тестового кода.

Ограничения или constraints

- Они нужны! Точка.
  - Отговорки не принимаются
  - Если они вам не нужны, то зачем вам транзакционное хранилище?
- Почему они нужны?
  - БД — это данные бизнеса
  - Ограничения — это способ избежать ошибок
  - Ошибки — это финансовые потери (или хуже)
  - Наблюдение: предложение самостоятельно оплачивать ошибки бизнеса действует на программистов просветляюще

Лучше не выполнить правильную операцию, чем выполнить неправильную

- Кстати: именуйте констрейнты. Это потом может пригодиться
- Кстати: проверок должно быть две. Если вы сломаете одну — спасет вторая

Опытный программист

- Где держать бизнес-логику?
  - В приложении? Ну ок (см. предыдущий слайд)
    - А если срочные правки?
  - В базе?
    - «Я пробовал, получилось еще неудобнее»
    - Опытный программист на фортране пишет на фортране на любом языке
    - create type user…
get_user_by_id(p_id integer) returns user
get_users_by_id(ids integer) returns user[]
…
Действительно черти что
    - Но зачем?! У вас РЕЛЯЦИОННАЯ СУБД.

Выводы

- Оценивайте важность данных
- Для важных данных **нужны констрейнты**
- Не надо писать процедурный код в реляционной БД. Это неудобно всем

Проектируем БД

- Не надо мудрить; хорошо спроектированная БД проста, логична, последовательная и потому понятна
  - Наблюдение: если у меня не очень хитрый бизнес, а база получается сильно хитрая, то я делаю что-то не то
  - Наблюдение: если я все время пишу хитрые запросы, то это не я такой умный, но у меня переусложненная/неудобная база
    - В счете указан адрес компании; через год компания меняет адрес; получается, что через год мы не можем напечатать счет годичной давности
    - Неправильное решение — вести историю адресов компании: мы получаем странный сервис «ведение неактуальной и ошибочной истории адресов других компаний, как они нам попадались»
    - Правильное решение — вести архив счетов
      - А кстати, как его вести?

Проектирование - пихаем все

- Чем меньше база, тем быстрее она работает и тем больше прощает ошибок
- Зато удобно – все вместе
- Если данных немного ИЛИ требуется транзакционный одновременный доступ — не раздумывая в базу
- Если данных много И не требуется транзакционный доступ — надо рассмотреть вопрос о выносе в отдельное хранилище — так и БД легче, и хранилище гибче
  - Автоматическая обработка .docx в хранимых процедурах выглядит отчасти странно
- Если данных мало — кладите все в базу, так удобнее
- Пример: БД, данных — 100Г, документов — 3Т. Все в базе
- Пример: БД, архив операций за 20 лет, толком требуется только несколько месяцев, по закону — 3 года; вместо нескольких терабайт получаем опять же около 100Г
- Наблюдение: если у вас не мегабизнес, а база получается большая, вы делаете что-то не то.

Проектирование — физические сущности

- Типичная ошибка: не надо бездумно добавлять индексы, они не бесплатны
  - Обратное тоже верно — не надо бояться индексов
- Типичная ошибка: секционирование жизненно необходимо при большом объеме данных
- Наблюдение — практически все большие БД — по большей части append-only

Проектирование - выводы

- «Военное искусство просто и вполне доступно здравому уму человека. Но воевать сложно» (Клаузевиц)
- Тем не менее:
  - Не надо мудрить
  - Не надо раздувать базу

Обработка ошибок

- Это самое страшное
- Почему-то очень многие считают, что все пойдет как надо, а если пойдет как не надо, то оно как-нибудь так… саморассосется
- Не саморассосется

Обработка ошибок — «да пропади оно все пропадом!»

- begin
  …
  exception when others then…
  - Null - самое классное!
  - Get stacked diagnostics
    ```
    error:=true;
    – еще что-то
    return;
    end;
    ```
- Если вы не знаете, что делать с ошибкой, то зачем тогда ее перехватываете?

Обработка ошибок — клиент

```
– (data, wasError) = doSomeOperation1();
  if wasError then
    log.error(«There was an error»….);
    doCleanup1();
    return ???;
  end if;
  … - тут, кстати, тоже могут быть ошибки
  (data, wasError) = doSomeOperation2();
  if wasError then
    log.error(«There was an another error»….);
    doCleanup2();
    return ???;
  end if;
```

Обработка ошибок - клиент

```
try{
    doSomeOperation1();
   doSomeOperation2();
}catch(AppException ex){
   doErrorHandling();
}finally{
   doCleanup();
}
```

Обработка ошибок - клиент

```
doSomeOperation1();
doSomeOperation2();
doSomeOperation3();
```

- SQLExceptionTranslator – Spring JDBC, set_exception_handler – php,...

Обработка ошибок - выводы

- Не перехватывайте ошибку, с которой не знаете, что делать
- Именуйте ошибки тщательно и аккуратно (см. выше про именование)
- Классифицируйте ошибки с **точки зрения приложения**: для приложения ошибка Connection refused и Deadlock detected по сути одна и та же(операцию можно и нужно повторить), а unique_violation и foreign_key_violation — могут быть совершенно разные.

Приложения

- Что там в приложении — ну, в общем, это не мое дело
  Я бы посоветовал что-то по мотивам исключений, а там как знаете
- Обратите внимание на именование ограничений — это может помочь при обработке ошибки
- Проблем в том, что так, кажется, мало кто делает
- Результат — idle in transaction
  - Наблюдение — для живых людей-операторов не требуется много соединений. Если требуется — вы делаете что-то не то.

Приложения — конкурентный доступ

- Потерянные обновления
  - Вася грузит форму, Петя грузит форму, Вася пишет длинный глубокомысленный комментарий и сохраняет форму, Петя просто жмакает «Сохранить» и через пять минут Вася видит, что его комментария нет. Дальнейшее развитие событий зависит от должности и характера Васи.
- Смех смехом, но мало кто на это обращает внимание
- Если люди хотя бы иногда одновременно работают над одной строкой в БД — надо проверять

Приложения — конкурентный доступ

- Ограничения!
  - Два вялых менеджера раз в неделю по очереди просматривали заявки; тем не менее они умудрились на одну заявку произвести две выплаты
- Вывод — если возможен конфликт на почве конкурентного доступа, то он рано или поздно случится

Ограничения!

Приложения - выводы

- Помните о конкуретности
- Помните об уровнях изоляции
- Констрейнты незаменимы

Неотменямые операции

- Begin transaction
  call_external_web_service(action:=do_payment...);
  …
- Проблемы:
  - дальнейшие попытки ее выполнить не предпринимаются - это еще неплохо
  - Предпринимаются! - а вот это совсем нехорошо

Неотменяемые операции

```
begin transaction
  update user set state='pending' [, processed_by=…] where state='ready' and id=...;
    if not found then
      raise sqlstate 'EW001' using message….
    end if;
commit;
begin transaction
    [ if exists(select * from user where id=… and state='pending' and processed_by=….
 then
        raise sqlstate 'EW002' using message=…
     end if;]
    call_external_web_service(action:=do_payment ...);
```

…

- Еще раз: **лучше не выполнить правильную операцию, чем выполнить неправильную**
- Давайте вспомним код выше с обработкой ошибок без try-catch-finally

  Ну да, можно представить

Неотменяемые операции - выводы

- Все операции делятся на четыре типа
  - В нашей бд (транзакционны, могут быть откачены)
  - Вне нашей бд, но в одной транзакции (2PC)
  - Вне нашей бд (нетранзакционны, идемпотентны)
  - **Вне нашей бд (нетранзакционны, неидемпотентны)**

Common Table Expression

- CTE не бесплатны
- CTE очень нравятся
- ОЧЕНЬ НРАВЯТСЯ
- ОЧЕНЬ!
- Надо быть аккуратным

Common Table Expression

- Общая проблема — дешевые lexically scoped view
- К сожалению, сейчас их нет.
- Что делать?
  - Параметризованные view/функции
  - Динамическое формирование запросов :-(

Postgres PROFESSIONAL

CTE - выводы

- CTE не бесплатны
- CTE — не средство логической организации запросов :-(
- Используйте view или функции

Увлечение JSON

- (id bigint primary key, val jsonb)
- А что? Работает же!
- Проблемы
  - Recheck
  - Index-only scan
- Пожалуй, так все-таки не надо делать

Postgres PROFESSIONAL

Вопросы?