Вот спойлер всех вопросов:

1. Нормализация
2. NoSQL
3. Транзакции
4. ACID
5. Выполнение транзакций
6. Аномалия сериализации
7. Уровни изоляции
8. Блокировки
9. Индексы
10. Планировщик
11. Оптимизация запросов
12. Масштабируемость
13. Что будет если сделать EXPLAIN ANALYZE DROP TABLE POSTGRES

1. Что такое "Нормализация"?

Есть такое понятие как *логическая избыточность*. Это когда, например, в таблице пользователей есть столбец номер телефона. То есть у пользователя Х может быть только один номер телефона для связи. Но зачастую это неудобно: ведь номеров у человека может быть несколько.

В такой ситуации разумно вынести номер телефона в отдельную таблицу и организовать связь "Один-ко-многим" (у одного пользователя может быть несколько телефонных номеров)

То есть мы разбиваем таблицу на части или делаем [декомпозицию](https://ru.wikipedia.org/wiki/%D0%9F%D1%8F%D1%82%D0%B0%D1%8F_%D0%BD%D0%BE%D1%80%D0%BC%D0%B0%D0%BB%D1%8C%D0%BD%D0%B0%D1%8F_%D1%84%D0%BE%D1%80%D0%BC%D0%B0#%D0%94%D0%B5%D0%BA%D0%BE%D0%BC%D0%BF%D0%BE%D0%B7%D0%B8%D1%86%D0%B8%D1%8F_%D0%B1%D0%B5%D0%B7_%D0%BF%D0%BE%D1%82%D0%B5%D1%80%D1%8C), приводя таким образом таблицу к нормальной форме

**Нормализация** как раз и подразумевает собой *процесс приведения базы данных к нормальным формам* с целью избавления от логической избыточности, а декомпозиция - это одна из вариаций нормализации.

Нормальных форм существует аж 8: с 1NF и до 6NF а также [Бойса-Кодда](https://ru.wikipedia.org/wiki/%D0%9D%D0%BE%D1%80%D0%BC%D0%B0%D0%BB%D1%8C%D0%BD%D0%B0%D1%8F_%D1%84%D0%BE%D1%80%D0%BC%D0%B0_%D0%91%D0%BE%D0%B9%D1%81%D0%B0_%E2%80%94_%D0%9A%D0%BE%D0%B4%D0%B4%D0%B0) и [Доменно-ключевая](https://ru.wikipedia.org/wiki/%D0%94%D0%BE%D0%BC%D0%B5%D0%BD%D0%BD%D0%BE-%D0%BA%D0%BB%D1%8E%D1%87%D0%B5%D0%B2%D0%B0%D1%8F_%D0%BD%D0%BE%D1%80%D0%BC%D0%B0%D0%BB%D1%8C%D0%BD%D0%B0%D1%8F_%D1%84%D0%BE%D1%80%D0%BC%D0%B0) формы.

2. Есть ли преимущество у NoSQL над SQL?

Иногда можно добиться большего быстродействия у первого языка. Если кратко - преимущество есть в скорости выполнения запросов. Это связано с отсутствием связей и конкретной схемы в NoSQL.

Так MongoDB может выигрывать у PostrgeSQL в запросах, которые подразумевают много связей и за которыми постгрес полезет в другие таблицы, которые, вдобавок, могут оказаться очень большими.

3. Что такое транзакция?

Транзакция - это элементарная операция в базе данных.

Однако транзакция может состоять и из нескольких операций: в этом ключе - это логически целостная процедура, в которой должны быть выполнены либо все операции - либо ни одна из них.

Транзакция начинается с команды BEGIN и заканчивается командой COMMIT либо отменяется командой ROLLBACK

4. Требования, предъявляемые к транзакции. ACID

Согласно теории баз данных транзакции должны обладать следующими свойствами:

1. **Атомарность** (***a****tomicity*). Это свойство означает, что либо транзакция будет зафиксирована в базе данных полностью, т. е. будут зафиксированы результаты выполнения всех ее операций, либо не будет зафиксирована ни одна операция транзакции.

2. **Согласованность** (***c****onsistency*). Это свойство предписывает, чтобы в результате успешного выполнения транзакции база данных была переведена из одного согласованного состояния в другое согласованное состояние.

3. **Изолированность** (***i****solation*). Во время выполнения транзакции другие транзакции должны оказывать по возможности минимальное влияние на нее.

4. **Долговечность** (***d****urability*). После успешной фиксации транзакции пользователь должен быть уверен, что данные надежно сохранены в базе данных и впоследствии могут быть извлечены из нее, независимо от последующих возможных сбоев в работе системы.

Для обозначения всех этих четырех свойств используется аббревиатура **ACID**.

5. Выполнение транзакций

Можно выполнять транзакции последовательно или параллельно

И если в первом случае все понятно, то со вторым могут возникать "феномены", например:

1. **Потерянное обновление** (*lost update*)

Когда разные транзакции одновременно изменяют одни и те же данные, то после фиксации изменений может оказаться, что одна транзакция перезаписала данные, обновленные и зафиксированные другой транзакцией.

2. **«Грязное» чтение** (*dirty read*)

Транзакция читает данные, измененные параллельной транзакцией, которая еще не завершилась. Если эта параллельная транзакция в итоге будет отменена, тогда окажется, что первая транзакция прочитала данные, которых нет в системе.

3. **Неповторяющееся чтение** (*non-repeatable read*)

При повторном чтении тех же самых данных в рамках одной транзакции оказывается, что другая транзакция успела изменить и зафиксировать эти данные. В результате тот же самый запрос выдает другой результат.

4. **Фантомное чтение** (*phantom read*).

Транзакция повторно выбирает множество строк в соответствии с одним и тем же критерием. В интервале времени между выполнением этих выборок другая транзакция добавляет новые строки и успешно фиксирует изменения. В результате при выполнении повторной выборки в первой транзакции может быть получено другое множество строк.

5. **Аномалия сериализации** (*serialization anomaly*)

Результат успешной фиксации группы транзакций, выполняющихся параллельно, не совпадает с результатом ни одного из возможных вариантов упорядочения этих транзакций, если бы они выполнялись последовательно.

6. Что такое аномалия сериализации?

Для двух транзакций, скажем, A и B, возможны только два варианта упорядочения при их последовательном выполнении:

1️⃣ сначала A, затем B

2️⃣ сначала B, затем A.

Причем результаты реализации двух вариантов могут в общем случае не совпадать.

Например, при выполнении двух банковских операций — *внесения некоторой суммы денег на какой-то счет* и *начисления процентов по этому счету* — важен порядок выполнения операций.

Если изначально на счете было 1000 у.е. и первой операцией будет *увеличение суммы* на 1000 у.е., а второй — начисление процентов (10%), то тогда итоговая сумма будет больше (2200 у.е.), чем при противоположном порядке выполнения этих операций (2100 у.е.).

Если описанные операции выполняются в рамках двух *различных* транзакций, то оказываются возможными *различные* итоговые результаты, зависящие от порядка их выполнения.

**Сериализация** двух транзакций при их параллельном выполнении означает, что полученный результат будет соответствовать одному из двух возможных вариантов упорядочения транзакций при их последовательном выполнении. То есть мы знаем, что результат может быть либо 2100 у.е. либо 2200 у.е. - не иначе. Соответственно, аномалия сериализации — несоответствие результата ни одному из вариантов.

При этом нельзя сказать точно, какой из вариантов будет реализован.

Если распространить эти рассуждения на случай, когда параллельно выполняется более двух транзакций, тогда результат их параллельного выполнения также должен быть таким, каким он был бы в случае выбора некоторого варианта упорядочения транзакций, если бы они выполнялись последовательно, одна за другой.

Конечно, чем больше транзакций, тем больше вариантов их упорядочения. Концепция сериализации не предписывает выбора какого-то определенного варианта. Речь идёт лишь об одном из них.

7. Уровни изоляции в SQL

1. *Read Uncommitted*

Это самый низкий уровень изоляции. Согласно стандарту SQL на этом уровне допускается чтение «грязных» (незафиксированных) данных.

Однако в PostgreSQL требования, предъявляемые к этому уровню, более строгие, чем в стандарте: чтение «грязных» данных на этом уровне не допускается.

2. *Read Committed*

Не допускается чтение «грязных» (незафиксированных) данных. Транзакция может видеть только те незафиксированные изменения данных, которые произведены в ходе выполнения ее самой.

3. *Repeatable Read*

Не допускается чтение «грязных» (незафиксированных) данных и неповторяющееся чтение. В PostgreSQL на этом уровне не допускается также фантомное чтение.

4. *Serializable*

Не допускается ни один из феноменов, перечисленных выше, в том числе и аномалии сериализации.

Конкретный уровень изоляции обеспечивает сама СУБД с помощью своих внутренних механизмов. Его достаточно указать в команде при старте транзакции.

Однако программист может дополнительно использовать некоторые операторы и приемы программирования, например, устанавливать блокировки на уровне отдельных строк или всей таблицы.

8. Блокировки

Кроме поддержки уровней изоляции транзакций, многие СУБД позволяют также создавать блокировки данных как на уровне отдельных строк, так и на уровне целых таблиц.

Команда SELECT имеет предложение FOR UPDATE, которое позволяет заблокировать отдельные строки таблицы с целью их последующего обновления.

Если одна транзакция заблокировала строки с помощью этой команды, тогда параллельные транзакции не смогут заблокировать эти же строки до тех пор, пока первая транзакция не завершится, и тем самым блокировка не будет снята.

Таким образом если выполнять данную команду:

**SELECT** \* **FROM** table\_tame **WHERE** column\_name ~ 'some text' **FOR** UPDATE;

на двух терминалах — сначала на одном — а затем на втором (с учетом начала транзакции BEGIN)

То можно заметить, что выполнение на втором терминале приостановится до тех пор пока не завершится транзакция первого терминала

При этом если на первом терминале выполнить какую-нибудь другую команду:

UPDATE table\_name **SET** column\_name = 'kek' **WHERE** column\_value = 404;

То, перейдя на второй терминал станет видно, что там была, наконец, выполнена выборка, которая покажет уже измененные данные с учетом данной UPDATE-команды

Ловушка джокера

9. Индексы

*Индекс* – специальная структура данных, которая связана с таблицей и создаётся на основе её данных. Индексы создаются для повышения производительности функционирования базы данных.

**Какие бывают индексы?**

* В-дерево
* хеш
* GiST
* SP-GiST
* GIN
* BRIN

По умолчанию команда CREATE INDEX создаёт индексы типа В-дерево (эффективны в большинстве случаев)

**Как можно создать индексы?**

🔹 Индекс по столбцу (это чистая классика)

🔹 Индекс по нескольким столбцам

🔹 Уникальный индекс

🔹 Индекс на основе выражения

🔹 Частичный индекс

Для создания индекса используется примерно такой синтаксис:

**CREATE** [**UNIQUE**] INDEX <index\_name> **ON** <table\_name> ( <column\_name>, ... ) [STATEMENT] ;

При этом:

для создания уникального индекса может использоваться слово **UNIQUE**

для создания выражения его записывают в скобках, например для создания выражения проверки индекса на нижний регистр можно написать так:

. . . ( lower( <column\_name> ) ) ;

для создания частичного индекса после скобок запись продолжается, например для проверки на величину можно написать так:

. . . ( . . . ) **WHERE** <column\_name> > 1000 ;

10. Планировщик

Это такой встроенный механизм в СУБД, перед выполнением запроса он формирует "план" выполнения запроса. Просмотреть план выполнения любого запроса можно с помощью команды EXPLAIN. Для детального понимания планов выполнения сложных запросов требуется опыт.

Планировщиком можно управлять с целью просмотра логики его работы и дальнейшей оптимизации запросов

11. Оптимизация запросов

Повлиять на скорость выполнения запроса можно различными способами. Я запомнил их так: изменить сам sql-запрос, обновить статистику планировщика, денормализация и 4 вариации изменения параметров планировщика:

* изменение исходного кода запроса
* обновление статистики, на основе которой планировщик строит планы
* денормализация: создание временных таблиц или создание индексов
* изменение параметров планировщика, управляющих выбором порядка соединения наборов строк
* изменение параметров планировщика, управляющих выбором метода доступа к данным (enable\_seqscan, enable\_indexscan, enable\_indexonlyscan, enable\_bitmapscan)
* изменение параметров планировщика, управляющих способом соединения наборов строк (enable\_nestloop, enable\_hashjoin, enable\_mergejoin);
* изменение параметров планировщика, управляющих использованием ряда операций: агрегирование на основе хеширования, материализация временных наборов строк, выполнение явной сортировки при наличии других возможностей.

12. Какие есть концепции масштабируемости БД

Я всегда выделял 2 основных. Это *Шардирование* и *Репликация.*В обоих случаях подразумевается наличие нескольких инстансов БД.

Для шардирования мы определяем срезы строк, которые будут храниться в конкретном инстансе, например для первого, это с 1 по 10 000, а для второго с 10 001 по 20 000 и так далее.

Репликация подразумевает создание одного инстанса-контроллера и нескольких инстансов-нод. То есть по сути это полное копирование СУБД на другой сервер. Таким образом контроллер сам будет решать в какую ноду и что ему записывать.

По статистике и, наверное, логике выборки данных SELECT в интернетах выполняются во много раз чаще, чем INSERT'ы, на этом фоне репликация выглядит привлекательнее.

Все выше сказанное относится к теме "Повышение отказоустойчивости". Вот пара статей про неё: [раз](https://vc.ru/u/1490572-free-chi76/625002-shardirovanie-vs-replikaciya-masshtabiruem-bd), [два](https://habr.com/ru/companies/oleg-bunin/articles/433370/)

13. Что будет если сделать EXPLAIN ANALYZE DROP DATABASE POSTGRES

Ну и напоследок такой вопрос с подвохом: всё плохо будет, потому что из-за ANALYZE субд мало того что с радостью удалит главную бд, так она ещё и посчитает веса а также время выполнения данного действа.

То есть простой EXPLAIN показывает план а EXPLAIN ANALYZE его ещё и выполняет

Заключение

Большую часть материала я брал из ~~интернетов~~ [книги Е.П. Моргунова "PostgreSQL. Основы языка SQL"](https://postgrespro.ru/education/books/sqlprimer)

