ACID vs BASE: копаем глубже баззвордов

ACID и BASE — это две аббревиатуры, которые предлагают удобную, но несколько упрощённую категоризацию систем (например, баз данных) и их гарантий. Они отражают более широкие философии проектирования, определяющие, как системы обрабатывают согласованность, доступность и надёжность.

Зачем писать о такой, казалось бы, простой теме?

Проблема, которую я хочу затронуть — это распространённое поверхностное использование этих терминов, которое проистекает из недостаточного понимания основ распределённых систем. Вместо использования их как баззвордов, я приглашаю вас присоединиться ко мне в анализе базовых идей, паттернов и проблем, лежащих в основе этого разделения.

- Правда ли, что ACID-базы данных плохо масштабируются?
- В чём разница между eventual consistency и weak consistency?
- Можем ли мы иметь ACID-транзакции в NoSQL БД? Могут ли реляционные БД предоставлять ослабленные (BASE) свойства согласованности?

Давайте разбираться!

Что такое ACID и BASE?

ACID: Фундамент транзакций

ACID фундаментально связан с транзакциями в базе данных. Термин был придуман в 1983 году в попытке определить терминологию для механизмов отказоустойчивости. Расшифровывается как:

Atomic (Атомарность)

Все транзакции выполняются по принципу "всё или ничего" (либо полностью завершаются, либо полностью откатываются — никаких частичных обновлений).

Consistent (Согласованность)

База данных гарантирует, что транзакции сохраняют все определённые инварианты — как явные (например, определенные constraint'ы), так и неявные (логика приложения).

Isolation (Изолированность)

Транзакции выполняются так, как будто они последовательны, даже когда идут параллельно. Уровни изоляции (например, Read Committed, Serializable) настраивают компромиссы между корректностью и производительностью.

Durable (Долговечность)

Данные сохраняются постоянно (даже после отключения сервера).

BASE: Альтернативный подход

Системы, которые не соответствуют этим критериям, называют BASE: Basically Available (В основном доступны), Soft state (Мягкое состояние) и Eventual consistency (Согласованность в конечном счёте).

Basically Available

Система остаётся работоспособной большую часть времени. Хотя некоторые части могут отказать, система в целом продолжает функционировать и обслуживать запросы.

Soft state

Состояние системы может изменяться со временем, даже без новых входных данных. Это происходит из-за eventual consistency — данные могут находиться в процессе изменения при передаче между узлами.

Eventual consistency

Система станет согласованной со временем, но не гарантирует немедленную согласованность между всеми узлами. Все узлы в конечном итоге придут к одному и тому же состоянию, но могут быть временные несоответствия.

Как BASE ослабляет ACID

BASE-системы могут ослаблять любое свойство ACID:

1. Атомарность → Частичное выполнение

   Возможно, операции будут выполнены только частично (например, "часть только часть инструкций будет выполнена успешно")

2. Согласованность → Временные нарушения

   Временные нарушения инвариантов (например, сумма балансов аккаунтов может кратковременно быть неверной)

3. Изолированность → Взаимное влияние

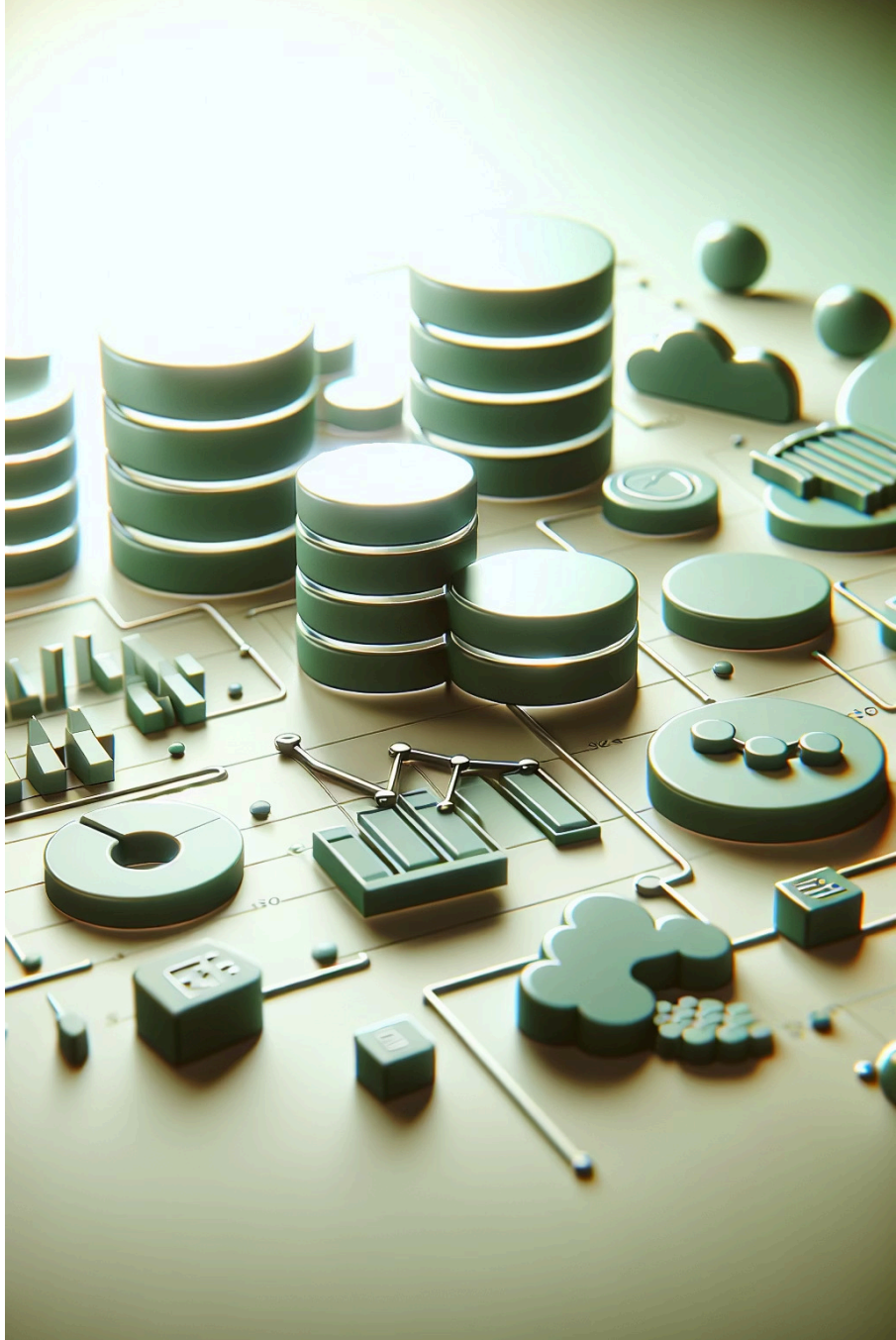
   Параллельные операции могут влиять друг на друга

4. Долговечность → Допустимые потери

   Допустимая потеря данных ради производительности (например, в распределенных кешах)

Миф ACID = Реляционные БД

Тут многие ошибаются: ACID не является эксклюзивным для реляционных баз данных, а Реляционные БД не всегда строго ACID.



NoSQL тоже может быть ACID

Несколько современных NoSQL баз данных предоставляют гарантии ACID:

MongoDB (4.0+)

поддерживает ACID-транзакции для нескольких документов

CockroachDB и YugabyteDB

предлагают распределённый SQL с полным соответствием ACID

Neo4j

(графовая база данных) предоставляет ACID-транзакции для операций с графами

Когда Реляционные БД становится BASE

Традиционные реляционные базы данных тоже могут быть BASE:

PostgreSQL / MySQL с репликами

Реплики для чтения и асинхронной репликацией: Реплики могут отставать от основной БД, обеспечивая eventual consistency для операций чтения

Шардированные Реляционных БД

Транзакции между шардами часто жертвуют строгой согласованностью ради производительности

Оптимизации для интенсивного чтения

В продакшене многие Реляционные БД дополняются слоями кеширования (Redis, Memcached), которые могут отдавать устаревшие данные

Почему появился BASE: Лимит масштабируемости

ACID звучит отлично, так зачем придумывать какой-то BASE? Ответ в том, что с ростом объёма и скорости данных реляционные БД начинают испытывать проблемы с производительностью и в какой-то момент не справляются с нагрузкой:

Пределы производительности РСУБД

100GB...

Размер БД

Проблемы с производительностью часто возникают, когда размер БД превышает этот порог

10K

RPS лимит

Реляционные БД обычно эффективно обрабатывают 1,000–10,000 RPS на среднем железе; свыше 10,000 RPS часто наблюдается деградация производительности

100M

Строк в таблице

Таблицы с 10–100 миллионами строк часто начинают показывать проблемы с производительностью

1K

Соединений

Большинство Реляционных БД эффективно обрабатывают не более 100–1,000 параллельных соединений

$10K

Стоимость масштабирования

Затраты на вертикальное масштабирование растут непропорционально: например, переход с 32 ядер на 64 может удвоить расходы (до примерно $10,000 в месяц), но даст только 1.4x производительности

Это тот момент, когда инженерам приходится рассматривать альтернативы традиционным РСУБД, которые поддерживают горизонтальное масштабирование.

BASE: размытое понятие

BASE — это не строгий стандарт; некоторые утверждают, что это больше маркетинговый баззворд. Как полушутя писал Мартин Клеппманн, лучшее определение BASE — это то, что это не ACID. BASE появился около 2008 года как альтернатива ACID, в основном из опыта таких компаний, как Amazon и eBay, которые уперлись в пределы традиционных реляционных баз данных.

> На самом деле BASE описывает следующее: когда у вас есть система, распределённая по нескольким машинам (возможно, между дата-центрами), теорема CAP говорит нам, что вы не можете иметь и идеальную доступность, и идеальную согласованность во время сетевых разделений. Поэтому эти системы выбирают доступность — они предпочтут дать вам слегка устаревший ответ, чем не дать ответа вообще.

На самом деле, многим приложениям не нужна строгая согласованность. Ваша лента ВКонтакте отстаёт на 2 секунды? Ну ок. В вашей корзине покупок есть небольшие несоответствия, которые исчезнуть через 100мс? Обычно это приемлемо. А как насчет банковский баланс? Тут нам нужны более сильные гарантии.

Так что BASE — это не совсем модель согласованности, это скорее **философия проектирования**, которая говорит: "Давайте будем прагматичны в требованиях к согласованности и оптимизируем вместо этого доступность и устойчивость к разделениям".

Спектр распределённой согласованности

Когда говорят о "согласованности", люди могут иметь в виду две разные вещи. "C" в ACID — это о поддержании инвариантов — баланс счёта никогда не становится отрицательным. Распределённая согласованность — это о том, когда изменения распространяются между узлами — именно это мы здесь рассматриваем.

Вместо того чтобы рассматривать ACID и BASE как две противоположности: strong consistency и eventual consistency, лучше смотреть на это как на спектр.

Модели распределённой согласованности

Linearizability ← → Eventual Consistency

1

Linearizability

Система выглядит так, как будто существует только одна копия данных, и все операции происходят атомарно в один момент времени. Как только клиент завершает запись, все клиенты, читающие из базы данных, должны видеть только что записанное значение.

2

Sequential Consistency

Все узлы видят операции в одном и том же порядке, и операции от каждого отдельного процесса появляются в том порядке, в котором они были выданы. Однако нет гарантий реального времени — узлы могут видеть операции со значительными задержками.

3

Causal Consistency

Если операция A причинно влияет на B, то каждый узел, который видит B, должен был ранее видеть A. Операции, которые не связаны причинно, являются параллельными и могут быть видны в разном порядке разными узлами. Фиксирует понятие того, что причины предшествуют следствиям.

4

Eventual Consistency

Крайне слабая — не говорит, когда произойдёт схождение или какие значения вы можете прочитать в промежутке. Может вернуть любое значение, которое когда-либо было записано.

Современные распределённые базы данных всё чаще предлагают настраиваемую согласованность (tunable consistency) — позволяя выбирать уровни согласованности для каждой операции. Amazon DynamoDB позволяет запрашивать строго согласованные чтения при необходимости. Эта гибкость представляет будущее: не ACID против BASE, а ACID, когда вам это нужно, BASE, когда нет. Или Google Spanner — он обеспечивает глобальную согласованность с разумной производительностью.

Выбор модели согласованности

Выбор между ACID и BASE зависит от сценария использования:

Выбирайте более сильную согласованность для:

- Финансовых транзакций и платежей
- Управления запасами с ограниченным количеством
- Аутентификации и авторизации пользователей
- Любого сценария, где несогласованность = бизнес-риск

Выбирайте более слабую согласованность для:

- Лент и таймлайнов в социальных сетях
- Рекомендаций товаров
- Аналитики и отчётности
- Доставки контента
- Любого сценария, где небольшая устарелость приемлема

Заключение

Ключевой вывод?

ACID и BASE — это не о типах баз данных, а о компромиссах. Понимание этих компромиссов, а не обращение с ними как с баззвордами — вот что позволяет нам создавать системы, которые балансируют согласованность, доступность и масштабируемость для наших конкретных потребностей