



# Базы данных

## Лекция 6

NoSQL: Введение



Мгер Аршакян

# План лекции

1

## **NoSQL: Концепции**

Когда реляционные БД перестают справляться с нагрузкой

2

## **Ограничения реляционных БД**

Ограничения реляционных БД в эпоху веб-масштабов

3

## **Основы распределённых систем: ACID vs BASE**

От строгой формальности к гибким гарантиям согласованности

4

## **Теорема CAP**

Что это такое?

5

## **Классификация NoSQL баз данных**

Key-Value, Document, Column-Family, Graph и Time-Series базы данных

6

## **Шардирование в NoSQL**

Стратегии разделения данных и решение возникающих проблем

7

## **Репликация в NoSQL**

Синхронная и асинхронная репликация, топологии и отказоустойчивость

8

## **Заключение и Q&A**

Подведение итогов и ответы на вопросы

# NoSQL: Концепции

1

## **NoSQL = "Not Only SQL"**

Изначально термин интерпретировался как "No SQL", но со временем эволюционировал в "Not Only SQL", подчеркивая дополительность, а не замену реляционных БД.

2

## **Отказ от "универсального решения" в пользу специализированных хранилищ**

Концепция "Универсальности" перестала работать, произошел переход от монолитных БД к экосистеме специализированных решений для конкретных задач.

3

## **Ориентация на горизонтальное масштабирование**

Способность линейно масштабироваться добавлением серверов вместо наращивания мощности.

4

## **Гибкость схемы данных при быстрой разработке продуктов**

Изменение схемы таблицы с миллиардами строк может занимать часы/дни в SQL.  
Схема-по-чтению в NoSQL позволяет развиваться быстрее при частых изменениях требований.

# Ограничения реляционных БД в эпоху веб-масштабов

1

## **Жёсткая схема данных**

ALTER TABLE в крупной БД требует блокировок. Частые изменения схемы создают операционные проблемы.

2

## **Join-операции**

$O(n*m)$  сложность при больших объёмах. При распределённых данных join между серверами часто невозможен.

3

## **Транзакции ACID**

В высоконагруженных системах длинные транзакции создают блокировки. Двухфазный коммит увеличивает задержки.

4

## **Высокая стоимость репликации в коммерческих РСУБД**

Oracle RAC (Real Application Clusters), MS SQL Server Enterprise Edition и т.д. требуют дополнительных лицензий а в NoSQL репликация обычно включена "из коробки".

# **ACID vs BASE**

**От строгой формальности к гибким  
гарантиям**

# Что такое ACID?

ACID — это набор из четырёх строгих свойств транзакций в СУБД:

- **Atomicity (атомарность):** либо все операции транзакции выполняются успешно, либо в случае ошибки откатываются все изменения.
- **Consistency (согласованность):** после завершения транзакции данные переходят из одного валидного состояния в другое, соблюдая все ограничения целостности.
- **Isolation (изолированность):** параллельные транзакции не влияют друг на друга, результаты выполняются так, будто они выполнялись последовательно.
- **Durability (долговечность):** после подтверждения транзакции её результаты сохраняются даже при сбоях системы или аппаратных сбоях.

# Что такое BASE?

BASE — это подход, противоположный ACID, ориентированный на высокую доступность и масштабируемость:

- **Basically Available (гарантированная доступность)**: система отвечает на все запросы, хотя данные могут быть частично или временно недоступны.
- **Soft-state (мягкое состояние)**: система допускает, что состояние данных может временно расходиться без внешнего запроса, благодаря фоновым процессам синхронизации.
- **Eventually consistent (согласованность в конце-концов)**: в отсутствие новых обновлений все реплики данных со временем придут к единому консистентному состоянию.

# Спектр компромиссов между согласованностью и доступностью

СУБД не разделяются строго на ACID или BASE: они располагаются на континууме, где разработчик выбирает баланс между двумя ключевыми свойствами:

1

## Жёсткая согласованность ↔ жёсткая доступность

ACID-системы склоняются к гарантированной согласованности даже ценой возможных простоев.

BASE-системы ставят доступность выше строгой согласованности, допуская рассогласование реплик.

2

## Настраиваемый уровень консистентности

**MongoDB** позволяет задать `write concern`, определяющий, сколько реплик должно подтвердить запись прежде чем она считается успешной.

**Cassandra** предлагает уровни «Consistency Level»: `ONE` (ответ от одного узла), `QUORUM` (большинство узлов) или `ALL` (все узлы).

3

## Гибридные подходы

Многие СУБД поддерживают ACID-транзакции внутри одной ноды и BASE-репликацию между нодами.

Можно комбинировать строгие транзакции для критичных операций и eventual consistency для вспомогательных данных (например, кэш или логи).



# Сравнительная таблица свойств ACID и BASE с примерами баз данных

Свойство	ACID	BASE
Доступность	Может быть ограничена ради согласованности	Всегда высокая
Согласованность	Строгая	Eventual
Масштабируемость	Вертикальная (ограничена одним узлом)	Горизонтальная (множество узлов)
Примеры БД	MySQL, PostgreSQL	Cassandra, CouchDB, DynamoDB

# Теорема CAP

## Свойства распределенных систем

### C (Consistency)

Согласованность: все узлы видят одинаковые данные в один момент времени.

Любое чтение возвращает результат последней записи или ошибку.

### A (Availability)

Доступность: каждый запрос успешно выполняется.

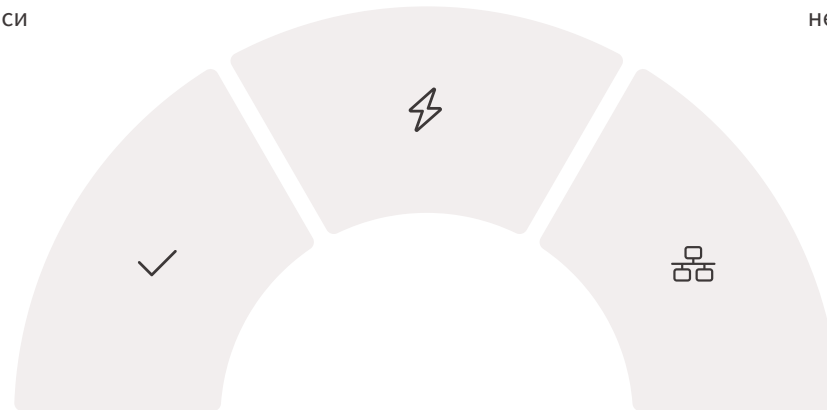
*Все работающие узлы возвращают ответ без ошибки, но не обязательно свежие данные.*

*Записывают данные.*

### P (Partition tolerance)

Устойчивость к разделению: система работает при сетевых разделениях.

В распределенных системах P — обязательное свойство, а не выбор. Система продолжает работать даже если между узлами нет связи.



1

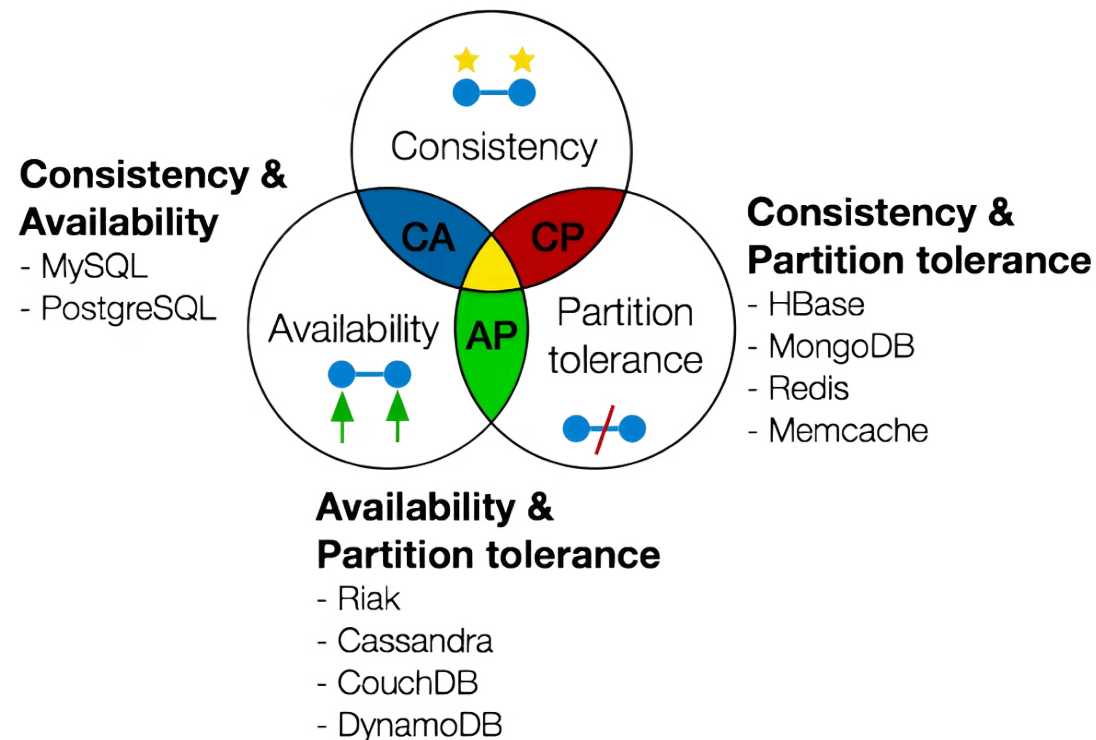
## Фундаментальное ограничение

Можно выбрать только 2 из 3 свойств (Доказанная математическая теорема Брюера (2000-2002), при сетевом разделении (P) приходится жертвовать либо C, либо A)

2

## Практический компромисс

**CAP** это выбор - в зависимости от задачи можно настраивать разные стратегии: *MongoDB* может настраивать баланс C/A через *write concern*, *Cassandra* — через уровни консистентности чтения/записи.



# Три доступные пары в CAP теореме

1

CA (Consistency + Availability)

2




CP (Consistency + Partition tolerance)

3

AP (Availability + Partition tolerance)

# CA (Consistency + Availability)

## Характеристики:

-  Согласованность: все узлы видят одинаковые данные одновременно
-  Доступность: каждый запрос получает ответ (успех или отказ)
-  Устойчивость к разделению: система не может работать при сетевых сбоях

## Примеры БД:




- MySQL
- PostgreSQL

## Особенности:

- Работают в рамках одного дата-центра
- При сетевом разделении система становится недоступной
- Идеально для классических ACID-транзакций
- Используют синхронную репликацию

# CP (Consistency + Partition tolerance)

## Характеристики:

-  Согласованность: данные всегда консистентны
-  Доступность: часть запросов отклоняется для сохранения согласованности
-  Устойчивость к разделению: система продолжает работать при сетевых сбоях

## Примеры БД:




- HBase
- MongoDB
- Redis
- Memcache

## Особенности:

- При сетевом разделении отдают приоритет консистентности
- Могут отклонять запросы для сохранения согласованности
- Используют алгоритмы консенсуса
- Подходят для критически важных данных

# AP (Availability + Partition tolerance)

## Характеристики:

-  Согласованность: допускается eventual consistency
-  Доступность: система всегда отвечает на запросы
-  Устойчивость к разделению: работает при сетевых проблемах

## Примеры БД:

- Riak
- Cassandra
- CouchDB
- DynamoDB

## Особенности:

- Приоритет отдается доступности над консистентностью
- Используют eventual consistency (согласованность в конечном счете)
- Отлично масштабируются горизонтально
- Подходят для высоконагруженных систем с географической распределенностью

# Eventual Consistency

## Компромисс в мире распределённых систем

1

**Определение: система достигнет согласованности в будущем**

Ключевое: **"eventually"**, нет гарантий времени, фокус на доступности

2

**Примеры: DNS, межбанковские переводы**

DNS с TTL, переводы минуты-дни

3

**Модели согласованности: сильная, слабая, причинная, монотонная, eventual**

**Сильная (Strong)** = атомарная видимость изменений всеми узлами, ACID-транзакции

**Слабая (Weak)** = без гарантий порядка или видимости, максимальная производительность, кэши

**Причинная (Causal)** = сохраняет порядок связанных событий, happens-before отношения

**Монотонная (Monotonic)** = никогда не видим "более старую" версию после "более новой", read/write monotonicity

**Eventual** = в итоге сходится без гарантий времени, высокая доступность, conflict resolution

4

**Конфликты и разрешение**

**Конфликт** = параллельные изменения одних данных, стратегии: **last-write-wins** (по временной метке), **merge** (слияние изменений), **custom resolution function**, **CRDTs** (Conflict-free Replicated Data Types).

5

**Техники: версионирование, quorum**

Версии изменений, большинство >50%



# **Классификация NoSQL баз данных**

# NoSQL: выбор правильного инструмента

1

## Специализация вместо универсальности (Polyglot persistence)

*Разные БД оптимизируются под разные нагрузки и паттерны доступа, пример — разные виды транспорта для разных задач (самолёт, автомобиль, велосипед), а не "универсальное транспортное средство"*

2

## Выбор модели данных

*Key-Value = быстрый точечный доступ, Document = гибкая структура и запросы, Column-Family = быстрая запись и анализ по колонкам, Graph = оптимальная работа со связями*

3

## No silver bullet

*Каждое решение имеет сильные и слабые стороны, невозможно одновременно оптимизировать все параметры (скорость, надёжность, гибкость, простота).*

# Key-Value хранилища: максимальная простота и скорость

1

## Модель данных

### Хеш-таблица в распределённом виде

*Простейшая модель "ключ → значение", значением может быть строка, число, JSON, бинарные данные*

2

## Примеры

*Redis, Memcached, etcd*

3

## Характеристики

Сверхбыстрые операции  $O(1)$

4

## Применения

### Кэширование, сессии, счётчики, очереди

*Кэш перед основной БД, пользовательские сессии, счётчики (просмотры, лайки); очереди и pub/sub (Redis), флаги, распределенные блокировки, реестр сервисов.*

5

## Ограничения

Примитивная структура, ограниченные возможности запросов

# Document-Oriented хранилища: гибкость + структура

1

## Модель данных

**Документ** - JSON/JSONB, **Коллекция** = набор документов (аналог таблицы)

2

## Примеры

MongoDB, Firebase

3

## Характеристики

Гибкая схема, индексация по любым полям, богатый язык запросов, агрегации

4

## Типичные применения

CMS, e-commerce, мобильные приложения

5

## Соответствие ООП

Документ соответствует экземпляру класса в ООП, нет необходимости в ORM-маппинге

# Column-Family хранилища: работа с большими данными

1

## Модель данных

Данные хранятся в виде колонок а не строк

2

## Примеры

ClickHouse

3

## Характеристики

Линейная масштабируемость, высокая производительность записи

4

## Применения

Аналитика, временные ряды, IoT, big data

5

## Ограничения

Сложность модели данных, денормализация

# Graph-базы хранилища: связи на первом месте

1

## Модель данных

Узлы, рёбра, свойства

2

## Примеры

Neo4j

3

## Характеристики

Природная работа со связями, оптимизация для обхода графа, запросы поиска пути и транзитивных связей, индексы для быстрого доступа к узлам и рёбрам

4

## Применения

Социальные сети, рекомендации, маршрутизация

5

## Преимущество

Операции со связями  $O(1)$

# Time-Series хранилища: данные с временной меткой

1

## Модель данных

Временные ряды, метрики, события

2

## Примеры

InfluxDB, Prometheus

3

## Характеристики

Высокая скорость записи, оптимизированные запросы по временным диапазонам

4

## Применения

Мониторинг, IoT, финансовые данные (цены акций, курсы валют), телеметрия оборудования

5

## Преимущество

Агрегация по временным интервалам (avg, min, max за час/день)

# **Шардирование в NoSQL**



# Шардирование: разделяй и властвуй

1

## Определение

Горизонтальное разделение данных между узлами. *Каждая партиция содержит полную схему, но подмножество записей, Распределение на разные физические серверы.*

2

## Цели

Увеличение ёмкости, улучшение производительности, *масштабируемость, локальность данных (приближение к пользователям)*

3

## Каждый шард — независимая БД с подмножеством данных

*Шард работает как полноценная БД*

4

## Прозрачность для приложения: маршрутизация запросов

*Маршрутизатор (router) определяет, к какому шарду обратиться, иногда клиент сам реализует маршрутизацию (Cassandra)*

# Стратегии шардирования: как разрезать данные

1

## **Range-based sharding: по диапазонам ключей (A-M, N-Z)**

*Правило разделения:* диапазоны значений ключа, пример - пользователи A-M на сервере 1, N-Z на сервере 2.

*Преимущества:* простота, эффективность диапазонных запросов.

*Недостатки:* неравномерность распределения (skew), горячие диапазоны.

2

## **Hash-based sharding: по хешу ключа**

*Правило разделения:* по хеш-функции к ключу для определения шарда.

*Преимущества:* более равномерное распределение.

*Недостатки:* отсутствие эффективности диапазонных запросов, только точечные.

3

## **Geo-based sharding: по географическому признаку**

*Правило разделения:* Распределение данных по географической близости к пользователям.

*Преимущества:* низкая латентность, соответствие законам о хранении данных.

*Недостатки:* сложный глобальный доступ.

4

## **Entity-based sharding: разные сущности на разных шардах**

*Правило разделения:* шардирование по типу данных; пример: пользователи на сервере A, заказы на сервере B.

*Преимущества:* позволяет оптимизировать для разных паттернов нагрузки.

*Недостаток:* сложности с запросами, требующими данные разных типов.

# Проблемы шардирования и их решения

1

## Неравномерное распределение данных (skew) и горячие точки

Skew = некоторые шарды содержат непропорционально больше данных, hot spot = шард с чрезмерно высокой нагрузкой, примеры - популярные пользователи.

2

## Объединение данных с разных шардов (cross-shard queries)

Запросы, требующие данные с разных шардов затратны, scatter-gather операция = запрос ко всем шардам + объединение результатов.

Варианты решения: denormalization, local joins в приложении, map-reduce для агрегации.

3

## Схемы ре-шардирования при росте данных

Процесс изменения количества шардов или правил распределения

Стратегии: offline (даунтайм), online (постепенная миграция)

4

## Глобальные операции: сортировка, агрегации, уникальность

Глобальные сортировки требуют данные со всех шардов - COUNT, SUM часто приблизительны, глобальные уникальные индексы сложны (нужна координация)

Стратегии: предагрегация на шардах, map-reduce, ограничение уникальности в рамках шарда, использование централизованного генератора ID

5

## Атомарные операции между шардами

Транзакции между шардами требуют 2PC (two-phase commit), что дорого, большинство NoSQL не поддерживают cross-shard транзакции

Решения: моделирование данных для избежания таких операций, локальные транзакции внутри шарда, компенсирующие транзакции

# Репликация в NoSQL

# Репликация: копии данных для надёжности и скорости

1

## **Определение: поддержание копий одних и тех же данных на разных узлах**

Дублирование данных между серверами для надёжности и скорости

Типы: репликация может быть полной (все данные) или частичной (только критичные)

Термины: копия = реплика, набор реплик = replica set

2

## **Цели: отказоустойчивость, производительность чтения**

Отказоустойчивость - работа при отказе отдельных узлов

Производительность - распределение нагрузки чтения (read scaling)

3

## **Вызовы: синхронизация копий, разрешение конфликтов**

Поддержание всех реплик в актуальном состоянии, обработка сетевых задержек, решение конфликтов при параллельных изменениях одних данных, обнаружение и восстановление потерянных или поврежденных данных, ресинхронизация после сбоев.

4

## **Компромисс: согласованность vs задержка репликации**

Strong consistency = высокие задержки записи (ожидание подтверждения от всех реплик)

Weak consistency = низкие задержки, но риск несогласованности

Спектр выбора: строгая согласованность критична для финансов, менее важна для социальных лайков.

5

## **Репликация + шардирование = полноценное масштабирование**

Шардирование = горизонтальное масштабирование ёмкости и пропускной способности записи

Репликация = надёжность и масштабирование чтения

# Стратегии репликации: синхронная vs асинхронная

1

## Синхронная: запись подтверждается после записи на все реплики

Клиент получает ответ только когда все реплики подтвердили запись  
Максимальная надежность (все копии идентичны)  
Минимальная доступность (блокировка при недоступности любой реплики)  
Высокая латентность (ожидание самой медленной реплики)

2

## Асинхронная: запись подтверждается до репликации на другие узлы

Запись подтверждается сразу после записи на мастер (primary), репликация на вторичные узлы — в фоне  
Минимальная латентность  
Максимальная доступность  
Риск потери данных при отказе мастера до репликации

3

## Полусинхронная: ждём подтверждения от части реплик (quorum)

Ожидание подтверждения от большинства ( $N/2+1$ ) узлов  
Баланс надежности и производительности - формула  $W+R>N$  для согласованности чтения/записи  
Настраиваемые уровни Consistency, (популярный практический выбор)

4

## Компромиссы: задержка vs надёжность vs доступность

Синхронно = высокая надежность + высокая латентность + низкая доступность  
Асинхронно = низкая надежность + низкая латентность + высокая доступность  
Golden mean = полусинхронность с quorum CAP-теорема определяет невозможность оптимизировать все три параметра одновременно.

5

## Практический выбор в зависимости от требований приложения

Финансы, платежи = синхронно (надежность критична)  
Социальные действия, логи = асинхронно (скорость важнее)  
E-commerce = quorum (баланс)

# Топологии репликации: master-slave и multi-master

1

## **Master-slave (primary-replica): один узел для записи, многие для чтения**

Архитектура с выделенным мастером для записи и репликами для чтения

Преимущества: простота, отсутствие конфликтов

Недостатки: единая точка отказа для записи, возможная перегрузка мастера

2

## **Multi-master: запись возможна на любой узел**

Все узлы равноправны, запись на любой

Преимущества: высокая доступность, географическое распределение записи

Недостатки: сложное разрешение конфликтов, потенциально более слабая согласованность

# Репликация и отказоустойчивость: обработка сбоев

1

## Обнаружение отказов: heartbeat, gossip protocol

Heartbeat: периодические сигналы "я жив" (типично 1-5 сек)

Gossip protocol: узлы обмениваются информацией о состоянии друг друга

Проблемы: false positives при временных сетевых задержках, настройка таймаутов (слишком короткие = ложные срабатывания, слишком длинные = медленная реакция)

2

## Выборы нового мастера (leader election)

Алгоритмы консенсуса для выбора нового лидера, Raft-подобный механизм голосования (большинство голосов)

3

## Сплит-брейн проблема и её предотвращение

Split-brain: два или более узла считают себя мастером, возникает при сетевом разделении  
Решения: кворумное голосование, требуется большинство  $(N/2+1)$



# Заключение

1

Нет универсального решения — выбор зависит от характера данных и запросов

2

Полиглот персистенс: комбинирование SQL и различных NoSQL подходов

# **Спасибо за внимание!**



**Беседа курса в  
Telegram**