

# Системы хранения и передачи данных: **Базы данных, их типы.**



Роман  
Гордиенко



## Роман Гордиенко

Backend Developer, Factory5



[Роман Гордиенко](#)

---

# Модуль «Системы хранения и передачи данных»

## Цели модуля:

- Узнать, чем различаются SQL и NoSQL базы данных, и научиться выбирать нужную из всего многообразия существующих решений;
- Разобраться, что такое кэш, зачем он нужен и как его использовать;
- Научиться складывать и смотреть логи приложений в Elasticsearch;
- Научиться настраивать и использовать менеджер очередей RabbitMQ.



---

# Структура модуля

1. Базы данных, их типы.
2. Кеширование Redis/memcached.
3. ELK.
4. Очереди RabbitMQ.





# План занятия

1. [Что такое Базы Данных?](#)
2. [Архитектурные многопользовательские модели СУБД](#)
3. [Системы Управления Базами Данных](#)
4. [Реляционная модель](#)
5. [Транзакции](#)
6. [Теоремы CAP и PACELC](#)
7. [NoSQL](#)
8. [Итоги](#)
9. [Домашнее задание](#)



# Что такое Базы Данных?



# Базы Данных

**Базы данных (БД)** — это структурная совокупность взаимосвязанных данных определённой предметной области: реальных объектов, процессов, явлений и т. д.

**База данных** — совокупность данных, организованных в соответствии с концептуальной структурой, описывающей характеристики этих данных и взаимоотношения между ними, которая поддерживает одну или более областей применения

---

# Базы Данных

Существует множество определений термина «*База Данных*» и из всех этих определений можно выделить ключевые моменты:

- БД хранится и обрабатывается в вычислительной системе;
- Данные в БД логически структурированы;
- БД включает схему или метаданные, описывающие её структуру.

Первый момент является строгим, а другие допускают различные трактовки и различные степени оценки.






## Базы Данных

Основная функция базы данных — предоставление *единого хранилища* для всей информации, относящейся к определённой теме или предметной области.

База данных может содержать всё, что угодно: будь то список приглашённых на свадьбу гостей или информация о каждом клиенте, посетившем web-сайт электронного магазина и разместившего там свои заказы.



# **Архитектурные многопользовательские модели СУБД**

---

# Архитектурные модели

Архитектурные модели разделяют на *три основных вида*, появление которых напрямую связано с развитием технологий и программного обеспечения:

- централизованная,
- файл — сервер,
- клиент — сервер.



## Централизованная

Система состоит из **центрального компьютера** и подключенных к нему **терминалов**.

Вся обработка данных выполняется в рамках этого единственного компьютера.

Пользовательские терминалы не обладают вычислительными мощностями и являются простым интерфейсом.

Центральный компьютер производит все вычислительные действия и обработку данных, передавая результат на терминалы.

## Файл — сервер

Система состоит из **файлового сервера**, который содержит файлы, необходимые для работы приложений и самой СУБД.

Пользовательские приложения и полные копии СУБД размещены и функционируют на отдельных рабочих станциях и обращаются к файловому серверу только по мере необходимости получения доступа к нужным им файлам.

Фактически, файловый сервер используется просто как общий жесткий диск.



## Файл — сервер

Поскольку файловый сервер не воспринимает команд на языке SQL, то СУБД должна запросить у файлового сервера файлы, необходимые для совершения запроса, что усложняет управление параллельной работой и контроль целостности данных, так как доступ к одним и тем же файлам могут осуществлять сразу несколько экземпляров СУБД.




## Клиент – сервер

Данная архитектура предполагает, что **СУБД находится на сервере**, и только она имеет доступ к файлам БД.

На клиентских компьютерах работают пользовательские приложения и клиентские компоненты СУБД, осуществляющие взаимодействие с сервером. От клиента на сервер приходят запросы, которые обрабатываются СУБД, и результат отправляется обратно клиенту.

Таким образом **повышается производительность системы**, гарантируется **согласованность и целостность данных**.

*Об этой модели более подробно поговорим позже, на примере PostgreSQL.*



# **Системы Управления Базами Данных**



---

# СУБД

**СУБД** — это программа, с помощью которой осуществляется хранение, обработка, управление и поиск информации в базе данных.

**СУБД** используются для выполнения различных операций с данными:

- Ввод,
- Хранение,
- Манипулирование,
- Обработка запросов к БД,
- Выборка,
- Сортировка,
- Обновление,
- Поиск,
- Защита данных от несанкционированного доступа или потери.

---

# СУБД

СУБД можно разделить на следующие категории:

- Реляционные
- Объектно-ориентированные
- NoSQL
  - Иерархические
  - Графовые
  - Сетевые
  - Документо-ориентированные
  - Ключ-значение
  - Column-oriented
- Протокол LDAP

# Реляционные СУБД

В системах управления реляционными базами данных отношения между данными являются реляционными, и данные хранятся в виде таблиц.

Каждый столбец таблицы представляет **атрибут**, а каждая строка в таблице представляет собой **запись**.

Каждое поле в таблице представляет собой **значение данных**.

Для взаимодействия с данными используют SQL .



The diagram shows a table titled 'employee'. A horizontal arrow labeled 'columns' points to the header row, and a vertical arrow labeled 'rows' points to the first column. The table has five columns: 'emp\_id', 'emp\_name', 'dep\_id', and two columns represented by ellipses. It has seven rows, with the first six rows containing data and the last row containing ellipses.

	emp_id	emp_name	dep_id	...
row 1	1	jack	1	...
row 2	2	ed	1	...
row 3	3	wendy	4	...
row 4	4	fred	4	...
row 5	5	sally	6	...
row 6	6	dogberg	8	...
...	...	...	...	...

# Объектно-ориентированные СУБД

Предоставляют  
полнофункциональные  
возможности программирования  
БД, сохраняя при этом  
совместимость с ООП языком.

Добавляет функциональность  
базы данных в ООП языки  
программирования.

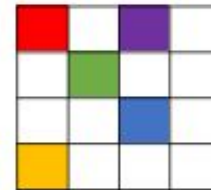


# NoSQL СУБД

Базы данных **NoSQL** не используют SQL в качестве основного языка доступа к данным.

NoSQL не имеет predetermined схем, что делает её идеальным кандидатом для быстро меняющихся сред разработки.

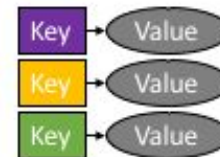
NoSQL позволяет разработчикам вносить изменения «на лету», не затрагивая приложения.



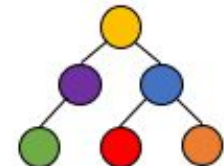
Column



Graph



Key-Value



Document

# Иерархические СУБД

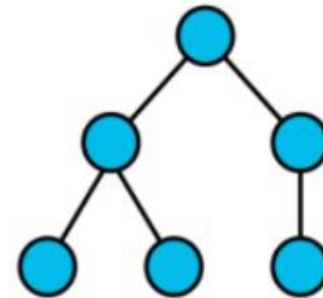
В **иерархической модели** данные организованы в древовидную структуру.

Данные хранятся в виде набора полей, где каждое поле содержит только одно значение.

Записи связаны друг с другом через связи в отношениях родитель-потомок.

В иерархической модели базы данных каждая дочерняя запись имеет только одного родителя.

Родитель может иметь несколько детей.



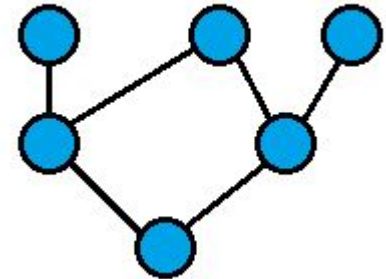
---

# Сетевые СУБД

**Сетевые СУБД** используют сетевую структуру для создания отношений между объектами.

Сетевые базы данных имеют иерархическую структуру, но в отличие от иерархических баз данных, где у одного дочернего элемента может быть только один родитель, сетевой узел может иметь отношения с несколькими объектами.

Сетевая база данных больше похожа на «паутину».



# Графовые СУБД

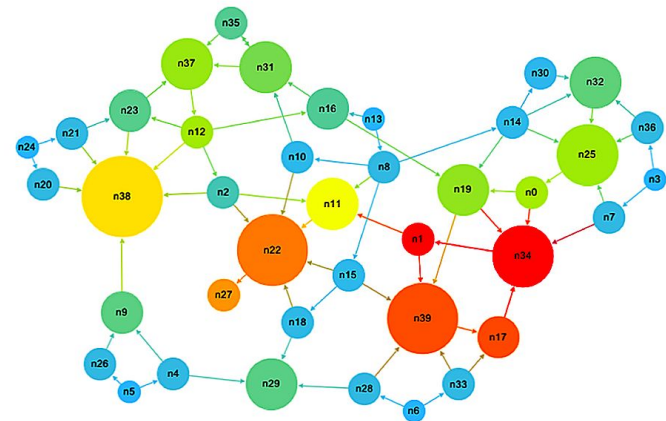
NoSQL БД, которая использует структуру графов для семантических запросов.

Данные хранятся в виде узлов, ребер и свойств.

Узел представляет собой объект.

Ребро представляет собой отношение, которое соединяет узлы.

Свойства — это дополнительная информация, добавляемая к узлам.





# Документо-ориентированные СУБД

Является NoSQL БД, в которой данные хранятся в виде документов.

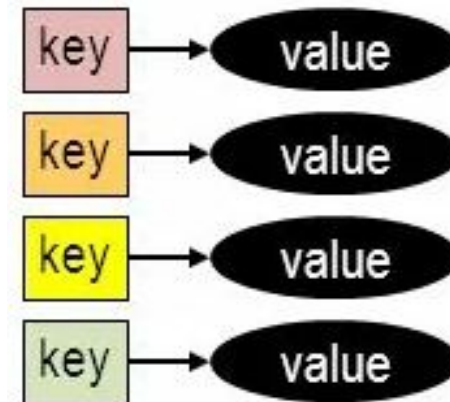
Каждый документ представляет данные в виде ключ-значение, связь с другими документами и мета-полями.

```
{  
  "name": "Bill",  
  "surname": "Gates",  
  "age": "48",  
  "company": {  
    "name" : "microsoft",  
    "year" : "1974",  
    "price" : "3000000"  
  }  
}
```

# Ключ-значение СУБД

База данных на основе пар «ключ-значение» хранит данные как совокупность пар «ключ-значение», в которых ключ служит уникальным идентификатором.

Как ключи, так и значения могут представлять собой что угодно: от простых до сложных составных объектов.

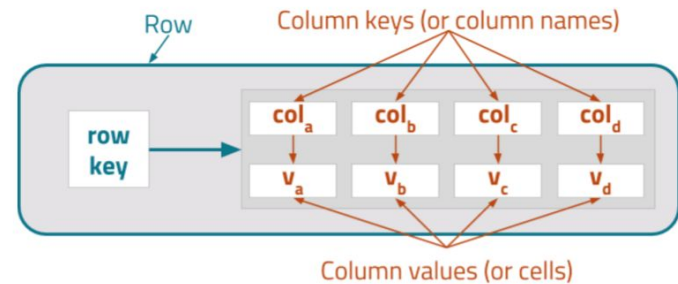


# Column – oriented СУБД

В таких системах данные хранятся в виде матрицы, строки и столбцы которой используются как ключи.

Типичным применением этого типа СУБД является веб-индексирование, а также задачи, связанные с большими данными, с пониженными требованиями к согласованности.

Каждая строка имеет свой набор столбцов.

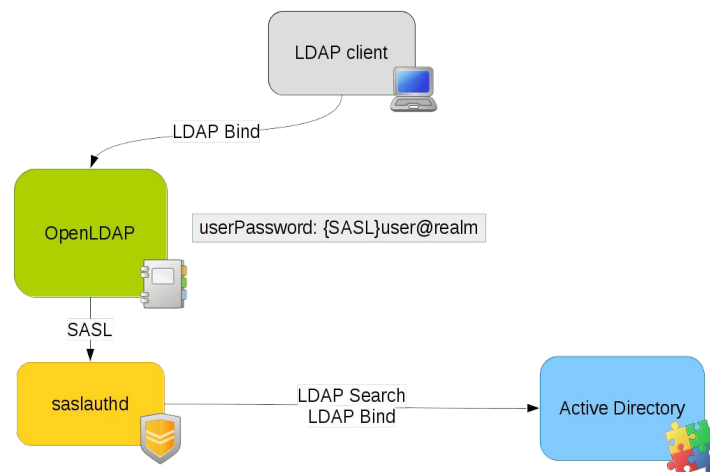


# LDAP СУБД

**Lightweight Directory Access Protocol** — это открытый и кроссплатформенный протокол, используемый для аутентификации служб каталогов.

LDAP — относительно простой протокол, использующий TCP/IP и позволяющий производить операции аутентификации (bind), поиска (search) и сравнения (compare), а также операции добавления, изменения или удаления записей.

Имеет преимущество в масштабируемых системах аутентификации перед РСУБД за счет каскадной репликации.





# Реляционная модель



## Реляционная модель

Модель представляет собой фиксированную структуру математических понятий, которая описывает, как будут представлены данные.

Базовой единицей данных в пределах реляционной модели является таблица.

**Таблица — это базовая единица данных.** В реляционной алгебре она называется «**отношение**» (relation). Состоит из атрибутов (columns), которые определяют конкретные типы данных. Данные в таблице организованы в кортежи (rows), которые содержат множества значений столбцов.

## Реляционная модель

Возвращаясь к *архитектурным моделям*, давайте на примере *PostgreSQL* разберем, из каких *подсистем* она состоит:

- **Клиентская часть** включает клиентское приложение и библиотеку *LIBPQ*, реализующую интерфейс связи с сервером. Библиотека *LIBPQ* отвечает за установление соединения с сервером и передачу SQL запросов.

---

## Реляционная модель.

- **Серверная часть** включает серверные процессы и контролирующий процесс-демон *Postmaster*, отвечающий за взаимодействие с клиентами.

*Postmaster* авторизует и принимает запросы от клиентов и осуществляет обмен данными между клиентом и сервером. При получении запроса соединения от клиента *Postmaster* создаёт соответствующий фоновый серверный процесс *postgres*, при этом используется связь один-к-одному. После того как серверный процесс создан, клиент и сервер взаимодействуют напрямую.





## Реляционная модель.

- **Хранение и управление данными** — несколько серверных процессов могут иметь одновременный доступ к информации из хранилища.

## Процессы и память

После того как клиент соединяется с сервером, процесс *postmaster* создает серверный процесс, и дальше клиент работает уже с ним. На каждое соединение создается по серверному процессу, поэтому при большом числе соединений следует использовать пул (расширение *pgbouncer*).

*Postmaster* также запускает ряд служебных процессов, увидеть которые можно с помощью команды *ps fax*.

У экземпляра базы данных имеется общая для всех серверных процессов память.



## Процессы и память

Большую часть памяти занимает **буферный кэш** (*shared buffers*), необходимый для ускорения работы с данными на диске.

*PostgreSQL* полностью полагается на операционную систему и сам не управляет устройствами.

Например, он считает, что вызов *fsync()* гарантирует попадание данных из памяти на диск.

---

# Процессы и память

Кроме буферного кэша в общей памяти находится **информация о блокировках и многое другое**, через нее же серверные процессы общаются друг с другом.

У каждого серверного процесса есть своя локальная память. В ней находится:

- кэш каталога (часто используемая информация о базе данных),
- планы запросов,
- рабочее пространство для выполнения запросов,
- и другое.



## Буферный кэш

Используется для оптимизации скорости работы памяти и дисков. Он состоит из массива буферов, которые содержат блоки данных и дополнительную информацию об этих блоках.

Размер блока обычно составляет 8 КБ, но может устанавливаться при сборке.

Буферный кэш, как и другие структуры в памяти, защищен блокировками от одновременного доступа. Блокировки организованы достаточно эффективно, чтобы не создавать большой конкуренции.

## Буферный кэш

Любой блок, с которым работает СУБД, попадает в кэш. Часто используемые блоки остаются в кэше надолго, редко используемые — вытесняются и заменяются другими блоками.

Буфер, содержащий измененный блок, называется «грязным».

Процесс *Background Writer* постепенно записывает такие блоки на диск в фоновом режиме, это позволяет снизить нагрузку на диски и увеличить производительность.

Если *Background Writer* не успевает записать вытесняемый серверным процессом грязный буфер, то процесс записывает его сам. С точки зрения производительности этого лучше не допускать.



# Кластеры

Кластер баз данных представляет собой набор баз, управляемых одним экземпляром работающего сервера.

После инициализации кластер будет содержать базу данных с именем *postgres*, предназначенную для использования по умолчанию утилитами, пользователями и сторонними приложениями.

Сам сервер баз данных не требует наличия базы *postgres*, но многие внешние вспомогательные программы рассчитывают на её существование.



## Кластеры.

Хранение данных на диске организовано с помощью табличных пространств.

Табличное пространство указывает расположение данных (каталог на файловой системе).

Оно может использоваться несколькими базами данных.

Например, можно организовать одно табличное пространство на быстрых дисках для активно использующихся данных и другое — на медленных дисках для архивных данных.





## Кластеры.

Объекты (таблицы и индексы) хранятся в файлах.

Каждый объект занимает один или несколько файлов внутри каталога табличного пространства. Кроме того, файлы разбиваются на части по 1 ГБ.

Необходимо учитывать влияние потенциально большого количества файлов на используемую файловую систему



# Транзакции

# Транзакции

**Транзакции** — это фундаментальное понятие во всех СУБД. Суть транзакции в том, что она объединяет последовательность действий в одну операцию **«всё или ничего»**.

Промежуточные состояния внутри последовательности не видны другим транзакциям, и если что-то помешает успешно завершить транзакцию, ни один из результатов этих действий не сохранится в базе данных.

Транзакции должны удовлетворять требованиям **ACID**: атомарность (*atomicity*), согласованность (*consistency*), изоляция (*isolation*) и долговечность (*durability*).

---

# Транзакции

- **Атомарность** означает, что при фиксации выполняются все операции, составляющие транзакцию, при откате — не выполняется ни одна.
- **Согласованность** — поддержание целостности данных. Транзакция начинает работу в согласованном (целостном) состоянии и по окончании своей работы также оставляет данные согласованными.
- **Изоляция** означает, что на результат работы транзакции не оказывают влияния другие, одновременно с ней выполняющиеся транзакции. По стандарту изоляция может иметь несколько различных уровней, в различной степени защищающих транзакции от внешних воздействий.
- **Долговечность** подразумевает возможность восстановить базу данных после сбоя в согласованном состоянии.



# Теоремы CAP и PACELC

---

# CAP

Теорема **CAP** (теорема *Брюера*) — эвристическое утверждение о том, что в любой реализации распределенных вычислений возможно обеспечить не более двух из трёх следующих свойств:

- согласованность данных (*consistency*),
- доступность (*availability*),
- устойчивость к разделению (*partition tolerance*).

---

# CAP

- **Согласованность данных (consistency)** — как только мы успешно записали данные в наше распределенное хранилище, любой клиент при запросе получит эти последние данные.
- **Доступность (availability)** — в любой момент клиент может получить данные из нашего хранилища или получить ответ об их отсутствии, если их никто еще не сохранял.
- **Устойчивость к разделению (partition tolerance)** — потеря сообщений между компонентами системы (возможно даже потеря всех сообщений) не влияет на работоспособность системы. Здесь очень важный момент: если какие-то компоненты выходят из строя, то это тоже подпадает под этот случай, так как можно считать, что данные компоненты просто теряют связь со всей остальной системой.

## Классы систем

В системе класса **CA** во всех узлах данные согласованы, и обеспечена доступность, при этом она жертвует устойчивостью к распаду на секции.

Система класса **CP** в каждый момент обеспечивает целостный результат и способна функционировать в условиях распада, но достигает этого в ущерб доступности: может не выдавать отклик на запрос.

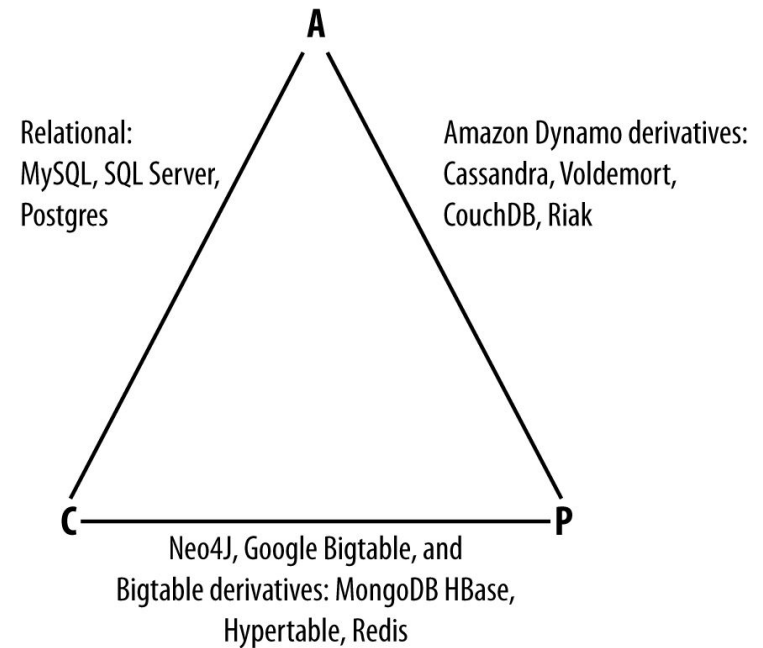
В системе класса **AP** не гарантируется целостность, но при этом выполнены условия доступности и устойчивости к распаду на секции. Хотя системы такого рода известны задолго до формулировки принципа *CAP* (например, распределённые веб-кэши или DNS).



# Классы систем

Несмотря на то, что РСУБД относят к **СА**, сложно представить системы не устойчивые к разделению. Как правило, приходится выбирать между **РА** и **РС**, в зависимости от бизнес-задач.

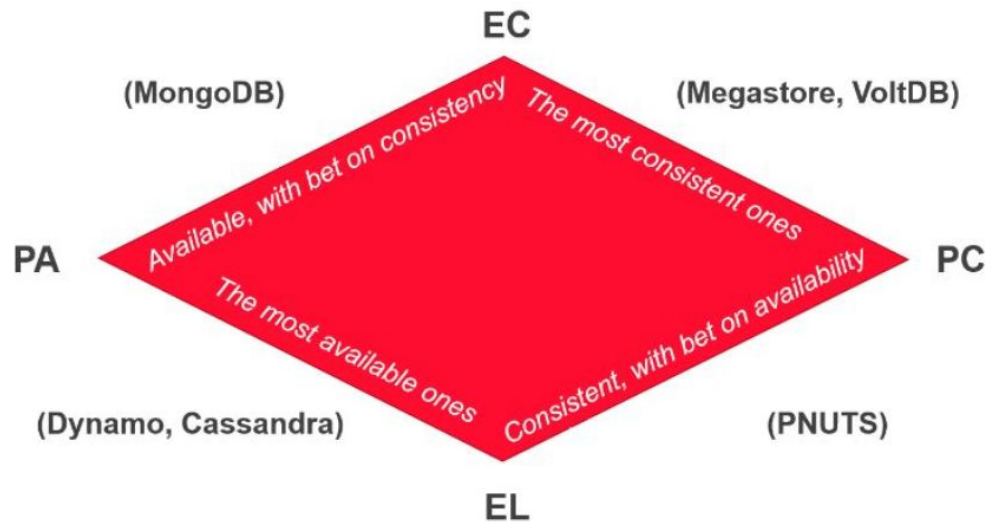
<https://codahale.com/you-cant-sacrifice-partition-tolerance/>



# PACELC

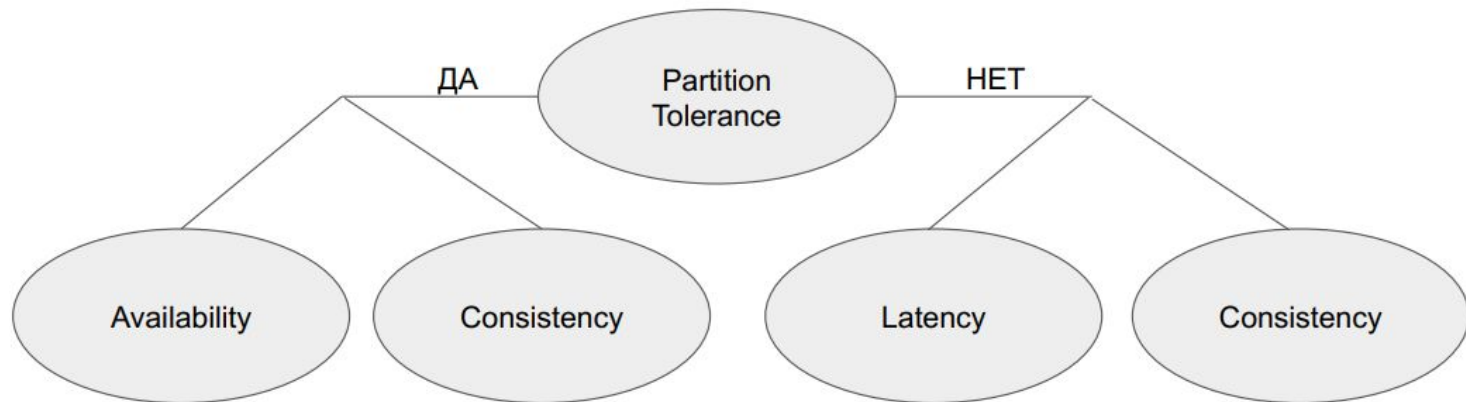
Расширение CAP-теоремы.

Добавляет понятие *Latency* — время, за которое клиент получит ответ и которое регулируется каким-либо уровнем согласованности.



# PACELC

В случае разделения сети (**P**) в распределённой системе необходимо выбирать между доступностью (**A**) и согласованностью (**C**) (согласно теореме CAP), но в любом случае, даже если система работает нормально в отсутствии разделения (**E**), нужно выбирать между задержками (**L**) и согласованностью (**C**)





# NoSQL

# BASE

**BASE** — принцип, противопоставляющий себя *ACID*.

- **BA** — *basically availability* (базовая доступность) деградация части узлов ведет к деградации части сессий, исключая полную деградацию системы. Система отвечает на любой запрос, но в ответе могут быть неверные данные.
- **S** — *soft state* (неустойчивое состояние) уменьшение времени хранения сессий и фиксация обновлений только критичных операций.
- **E** — *eventually consistent* (конечная согласованность) изменение состояния в конечном итоге применится на все системы.

BASE позволяет проектировать высокопроизводительные системы.

---

# Преимущества NoSQL

- Гибкость
- Масштабируемость
- Высокая производительность
- Широкие функциональные возможности

# NoSQL. MongoDB

**MongoDB** — одна из популярных документо-ориентированных СУБД. **MongoDB** поддерживает:

- ad-hoc запросы,
- индексирование,
- горизонтальное масштабирование и шардинг,
- MapReduce,
- транзакции, ACID/BASE.

По PACELC теореме MongoDB соответствует PA/EC.

# NoSQL. MongoDB

Модель устройства базы данных в **MongoDB**:

- База данных состоит из коллекций;
- Центральным понятием является документ;
- Документ представляет набор пар ключ-значение.





# NoSQL. MongoDB.

Выборка из БД, сравнение с SQL.

```
db.users.find({name: "Tom"})
```

```
select * from users where name = 'Tom'
```

```
db.users.find({$or : [{name: "Tom"}, {age: {$gte:30}}]})
```

```
select * from users where name = 'Tom' or age >= 30
```



# Итоги

---

# Итоги

## Сегодня мы:

- узнали, что такое Базы Данных;
- рассмотрели типы СУБД и архитектурные многопользовательские модели СУБД;
- познакомились с реляционными моделями, транзакциями и теоремами CAP / PACELC;
- разобрали отличие ACID и BASE принципов;
- узнали о NoSQL и MongoDB в частности.





# Домашнее задание

---

# Домашнее задание

Давайте посмотрим ваше [домашнее задание](#).

- Вопросы по домашней работе задавайте **в чате** мессенджера Slack.
- Задачи можно сдавать **по частям**.
- Зачёт по домашней работе проставляется после того, как **приняты все задачи**.

**Задавайте вопросы и  
пишите отзыв о лекции!**

**Роман Гордиенко**