# C# Professional

## Базы данных: NoSQL базы и их особенности

otus.ru


**Проверить, идет ли запись**

# Меня хорошо видно && слышно?

Ставим “+”, если все хорошо
“-”, если есть проблемы

Тема вебинара

# Базы данных: NoSQL базы и их особенности


**Михаил Дмитриев**

**Ведущий программист НИПК Электрон**

Разрабатываю и поддерживаю приложения для работы с радиологическими комплексами

https://t.me/sf321

# Правила вебинара

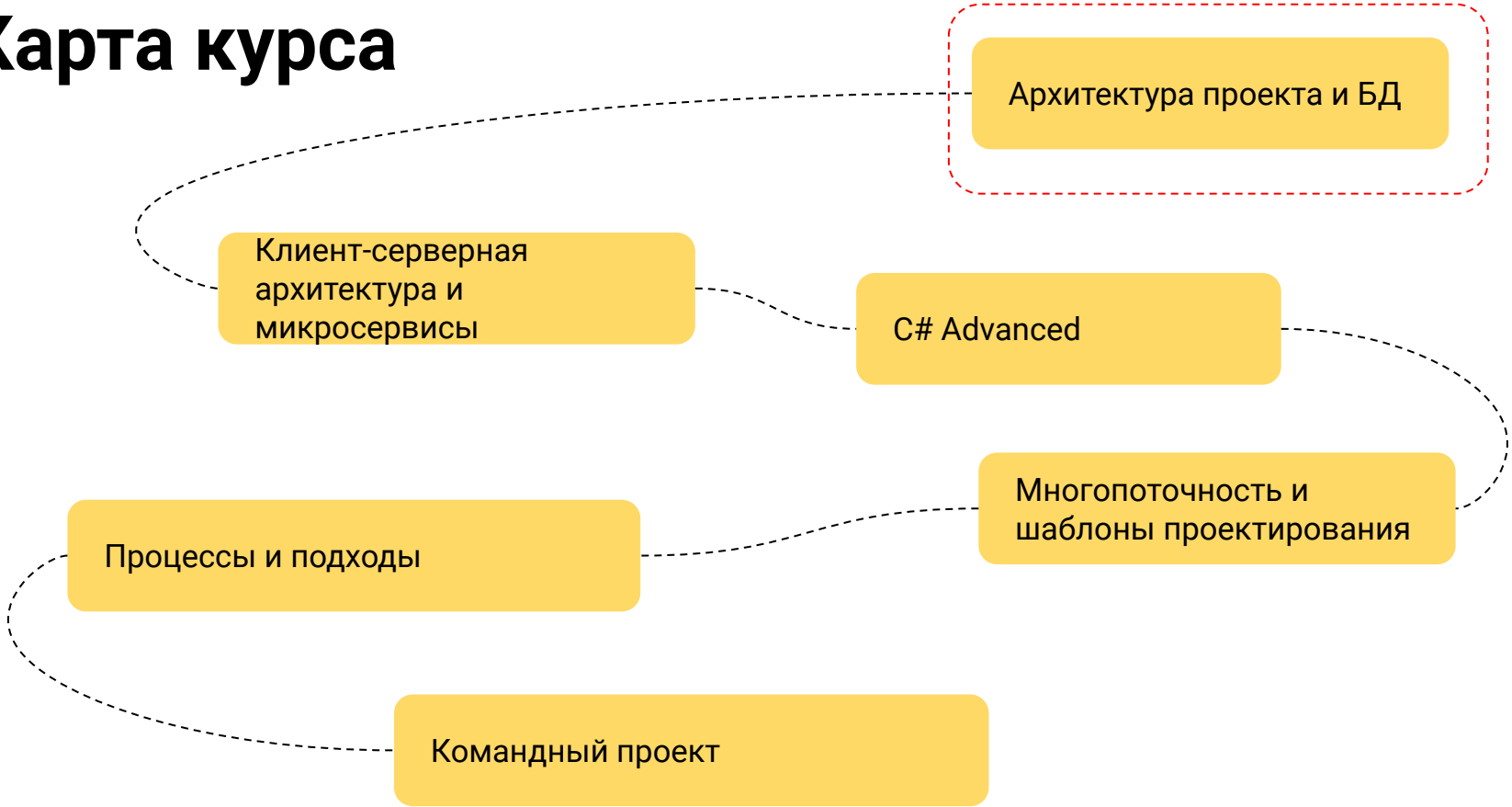
Активно
участвуем

Off-topic обсуждаем
в общем чате учебной группе
в telegram

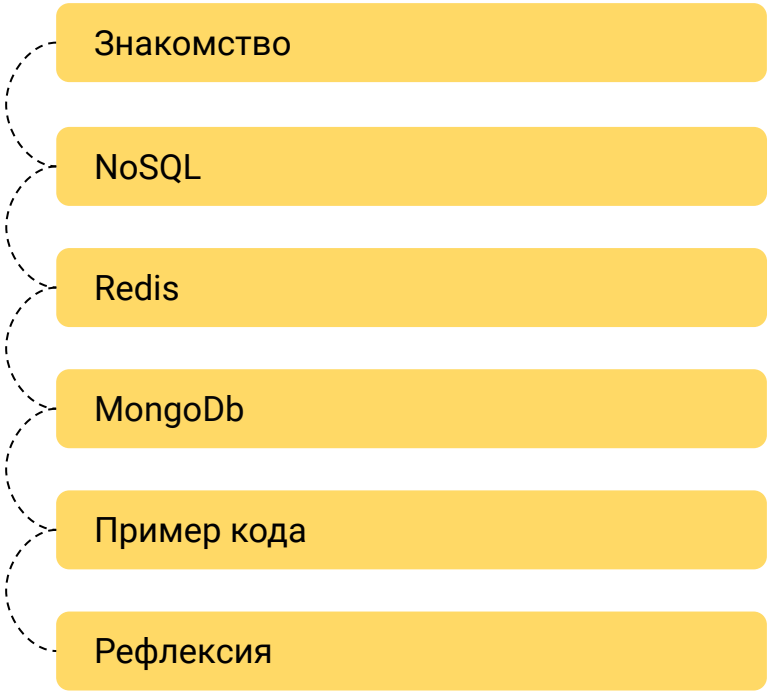
Задаем вопрос
в чат или голосом

Вопросы вижу в чате,
могу ответить не сразу

# Карта курса


Архитектура проекта и БД
Клиент-серверная архитектура и микросервисы
C# Advanced
Многопоточность и шаблоны проектирования
Процессы и подходы
Командный проект

# Маршрут вебинара



# Цели вебинара

## К концу занятия вы сможете

1. Различать SQL и NoSQL субд
2. Знать основные виды NoSQL субд и основных представителей этих видов
3. Понимать по каким критериям выбирается NoSQL субд для проекта
4. Использовать некоторые NoSQL субд в своих проектах

# Смысл

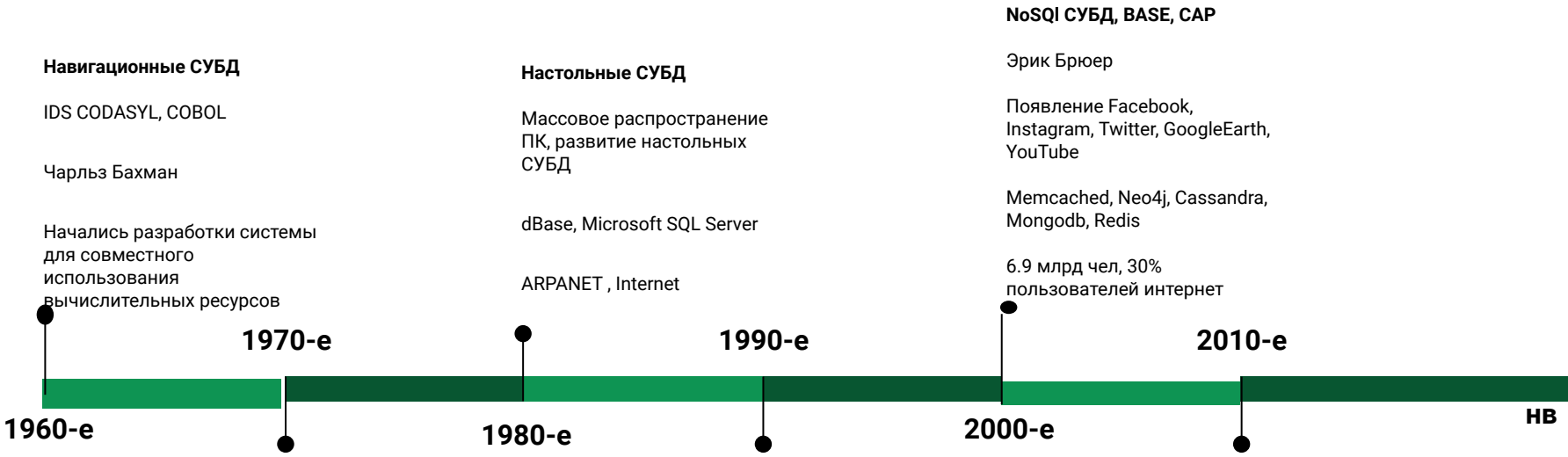
## Зачем вам это уметь

1. Понимать для решения каких задач имеет смысл использовать NoSQL бд
2. Подключать и работать в своем проекте с некоторыми NoSQL бд
3. Расширить ваш кругозор на вопросы организации хранения и обработки данных

# Тестирование

# NoSQL

# От реляционных СУБД к NoSQL СУБД




**Навигационные СУБД**

IDS CODASYL, COBOL

Чарльз Бахман

Начались разработки системы для совместного использования вычислительных ресурсов

**Реляционные СУБД, ACID**

Эдгар Кодд, Джим Грей

IBM Db2 SystemR, Oracle, Postgre

**Настольные СУБД**

Массовое распространение ПК, развитие настольных СУБД

dBase, Microsoft SQL Server

ARPANET , Internet

**Объектно ориентированный подход в работе с данными**

Объектно-реляционные базы данных, ORM

Google, Amazon, Ebay

**NoSQl СУБД, BASE, CAP**

Эрик Брюер

Появление Facebook, Instagram, Twitter, GoogleEarth, YouTube

Memcached, Neo4j, Cassandra, Mongodb, Redis

6.9 млрд чел, 30% пользователей интернет

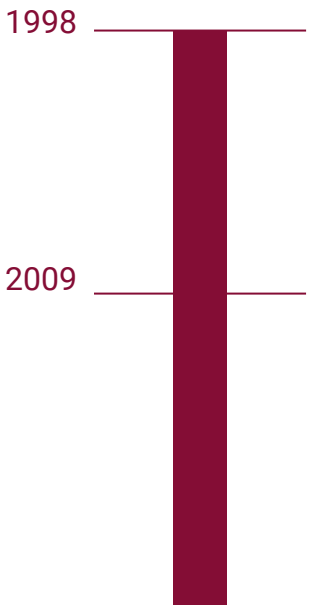
**PACELC , NewSQl СУБД**

Даниэль Дж. Абади

7.75 млрд чел 63% пользователи Интернет

5.6 млрд мобильных устройств

# Интересные факты о термине NoSQL




**NoSQL впервые прозвучал**

Carlo Strozzi назвал так свою реляционную базу данных, в которой вообще не пользовался SQL для работы с данными

**NoSQL как тип СУБД**

на конференции для обсуждения новых распределенных нереляционных баз данных с открытым исходном кодом был снова введен термин NoSQL. NoSQL появился как хэштег в твиттере

Новая трактовка — это «Not only SQL» — это не только SQL, т.е. может присутствовать и SQL, и дополнительные механизмы работы с данными

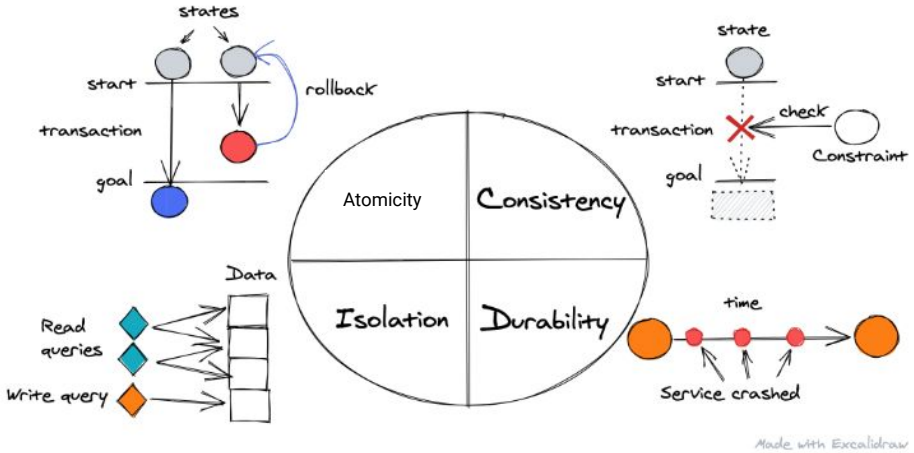
NoSQL стал общим термином для различных баз данных и хранилищ, но он не обозначает какую-либо одну конкретную технологию или продукт.

# Сравнение SQL и NoSQL СУБД

| | SQL | NoSQL |
|---|---|---|
| СУБД | реляционная | не-реляционная или распределенная |
| Схема данных | Зафиксированная или статическая предопределенная | динамическая |
| Хранилище данных | БД не ориентирована на иерархическое, распределенное хранилище данных | БД ориентированы на иерархическое, распределенное хранилище данных |
| Масштабируемость | вертикальная | горизонтальная |
| Принципы построения | ACID | BASE |

# ACID принципы в транзакционной системе

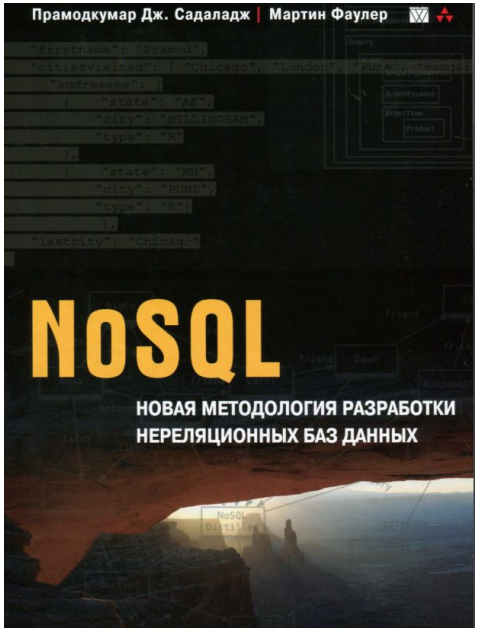
- **Atomicity - атомарность** транзакций
- **Consistency - согласованность** данных базы после завершения транзакций
- **Isolation - изолированность** транзакций
- **Durability - долговечность** данных, независимо от внутренних или внешних сбоев



# BASE принципы работы NoSQL СУБД

- **Basically available- базовая доступность** - каждый запрос гарантированно завершается (успешно или безуспешно)
- **Soft state- гибкое состояние** - состояние системы может меняться с течением времени, даже без ввода новых данных
- **Eventually consistent**- данные могут быть некоторое время рассогласованы, но приходят к согласованному состоянию через некоторое конечное время

# NoSQL СУБД




Martin Fowler “NoSQL Distilled: A Brief Guide to the Emerging World of Polyglot Persistence”


Eric Evans “Domain Driven Design”

# Паттерн “Агрегат”

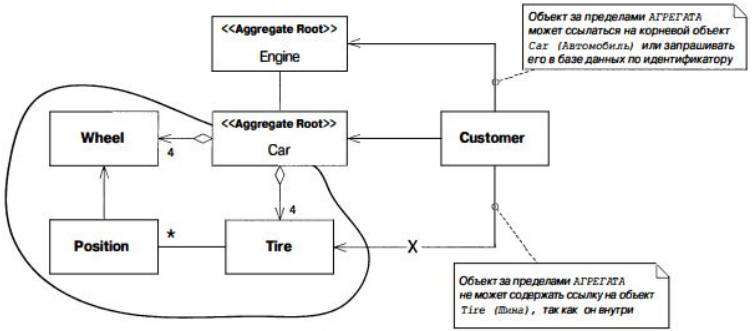



*Рис. 6.2. Локальная/глобальная идентичность и ссылки на объекты*

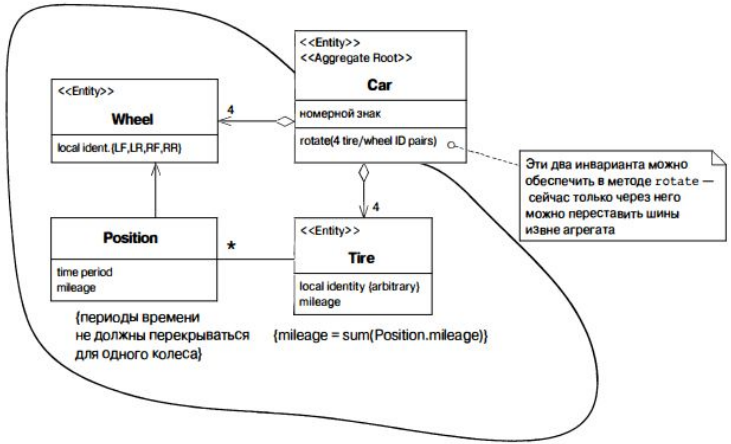
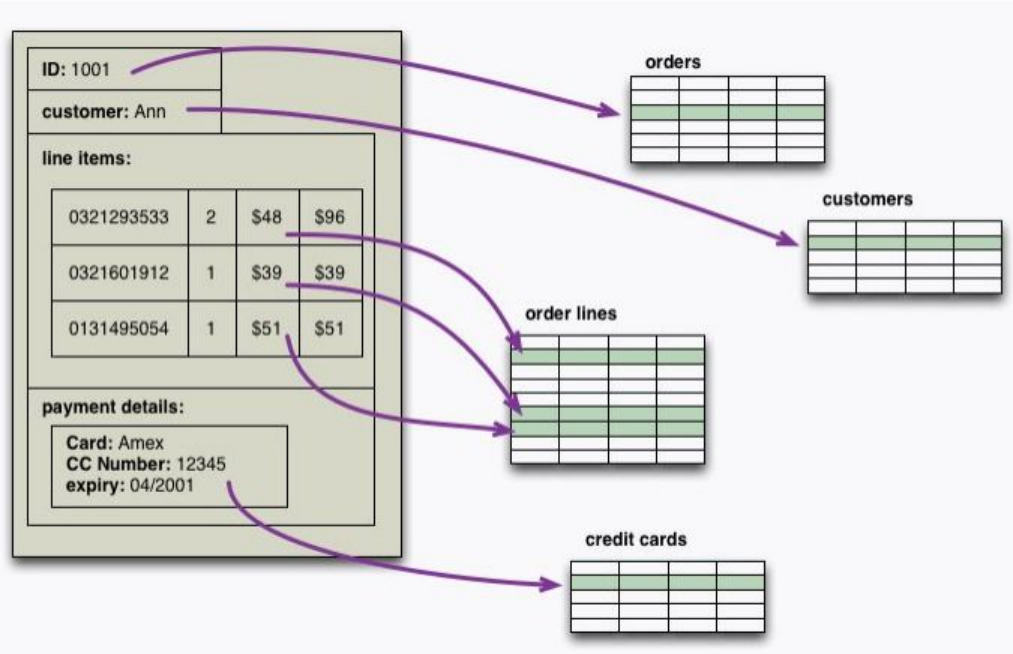



*Рис. 6.3. Инварианты АГРЕГАТОВ*

Совокупность взаимосвязанных объектов, которые мы воспринимаем как единое целое с точки зрения изменения данных, называется АГРЕГАТОМ (AGGREGATE). У каждого АГРЕГАТА есть **корневой объект** и есть **граница**. Граница определяет, что находится внутри АГРЕГАТА. Корневой объект - это один конкретный объект - СУЩНОСТЬ (ENTITY), содержащийся в АГРЕГАТЕ. Корневой объект - единственный член АГРЕГАТА, на который могут ссылаться внешние объекты, в то время как объекты, заключенные внутри границы, могут ссылаться друг на друга как угодно. СУЩНОСТИ, отличные от корневого объекта, локально индивидуальны, но различаться они должны только в пределах АГРЕГАТА, поскольку никакие внешние объекты все равно не могут их видеть вне контекста корневой СУЩНОСТИ.

# Паттерн “Агрегат”

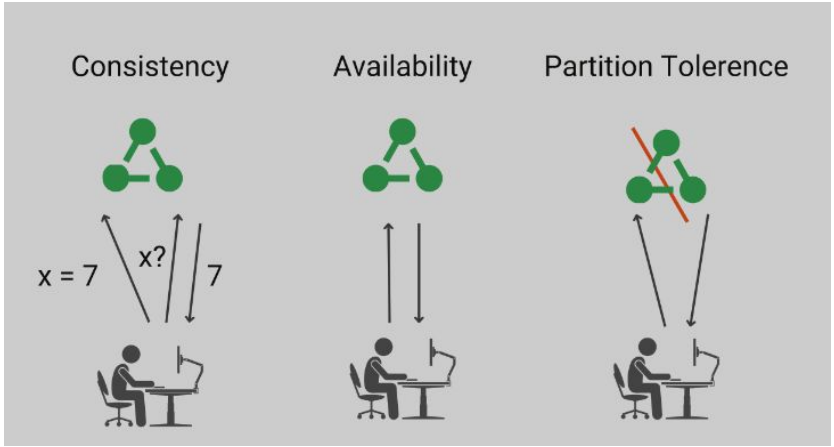



- Единая сущность в доменной модели
- Поиск данных по ключу (возможно через индекс)
- Единица хранения данных
- Простота ORM

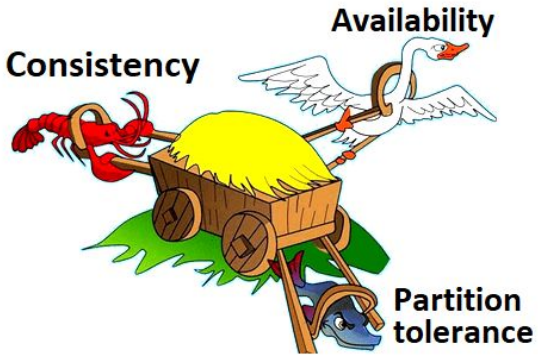
# CAP теорема Эрика Брюера

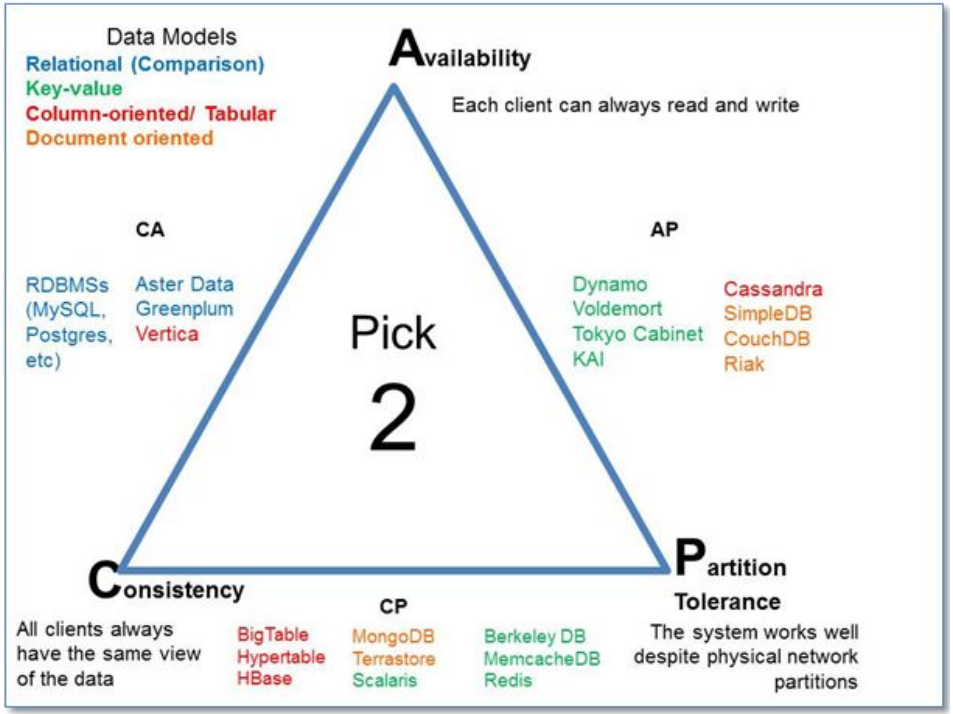
Эвристическое утверждение о том, что в любой реализации распределённых вычислений возможно обеспечить не более двух из трёх следующих свойств:

- **согласованность данных (consistency)** — во всех вычислительных узлах в один момент времени данные не противоречат друг другу
- **доступность (availability)** — любой запрос к распределённой системе завершается корректным откликом, однако без гарантии, что ответы всех узлов системы совпадают
- **устойчивость к разделению ( partition tolerance)** — расщепление распределённой системы на несколько изолированных секций не приводит к некорректности отклика от каждой из секций.



# CAP теорема Эрика Брюера

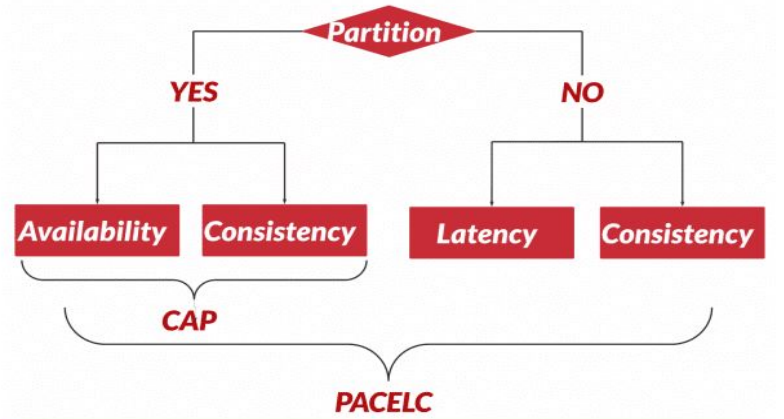


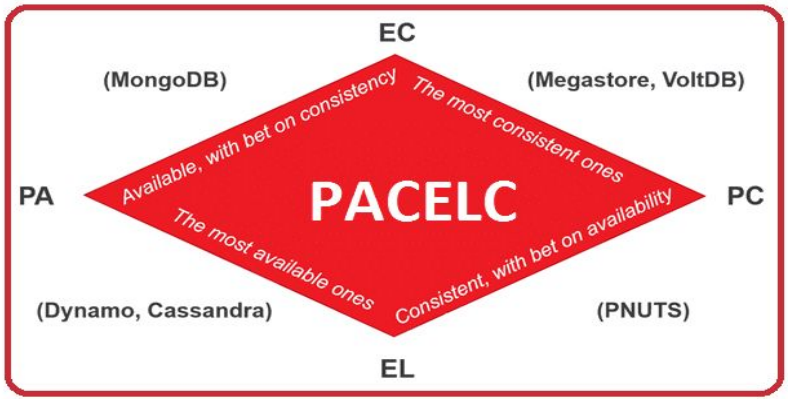



# PACELC теорема - расширение CAP теоремы

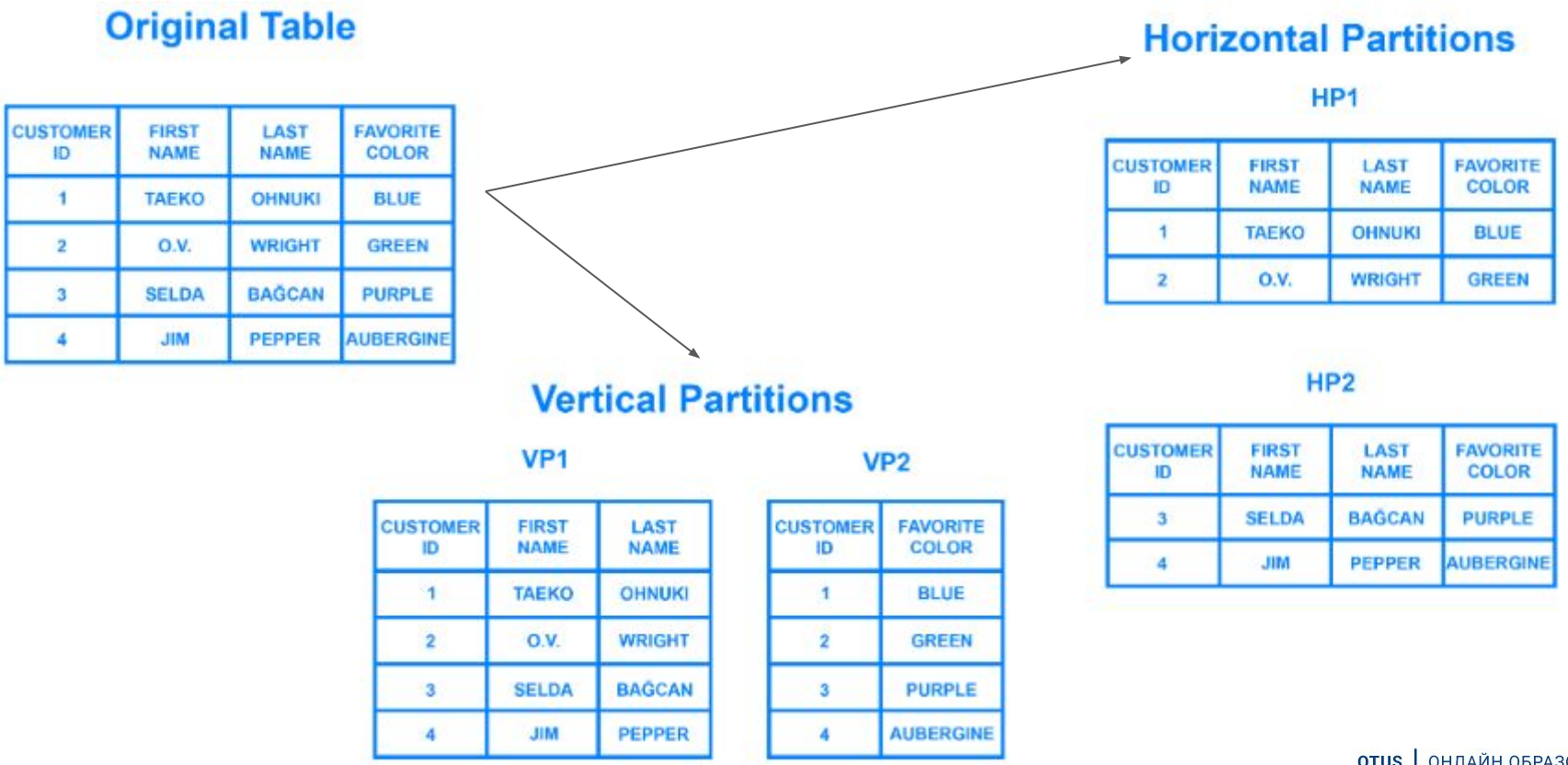
В случае разделения сети (P) в распределенной компьютерной системе необходимо выбирать между доступностью (A) и согласованностью (C) (согласно теореме CAP), но в любом случае, даже **если система работает нормально в отсутствии разделения** (E), нужно выбирать между :

- **задержками (Latency)**
- **согласованностью (Consistency)**

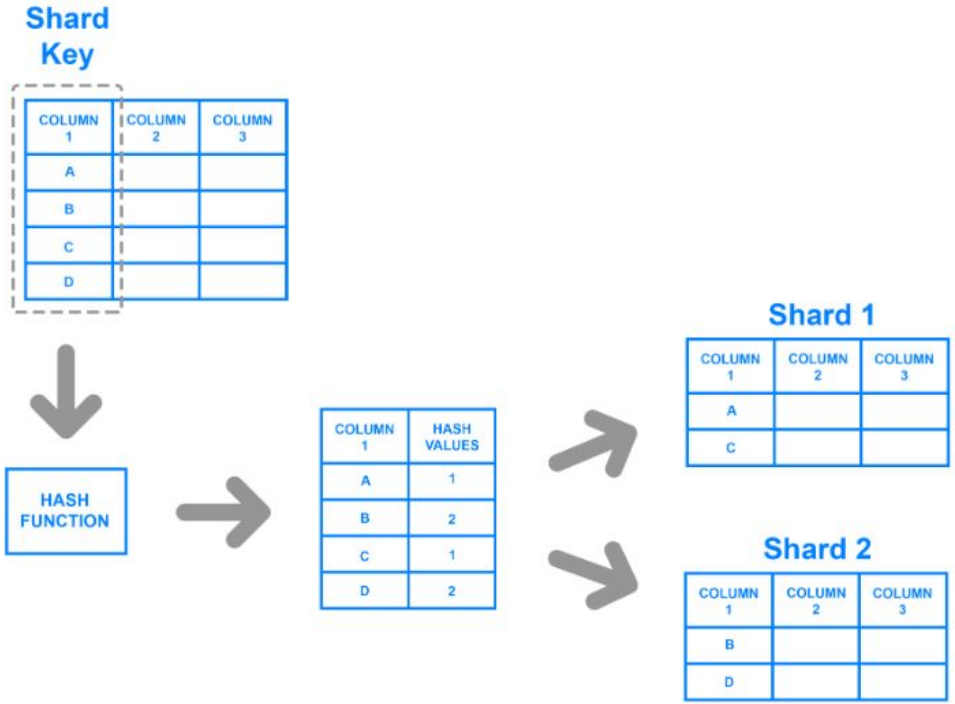


# Фрагментация

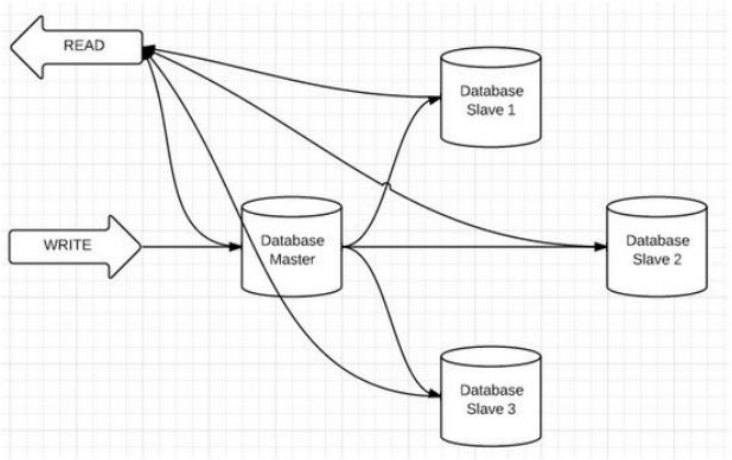


# Фрагментация

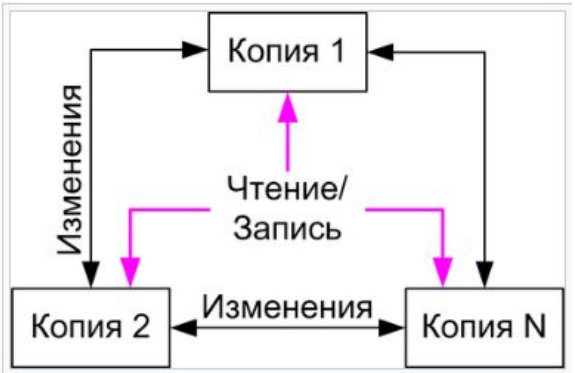


# Репликации

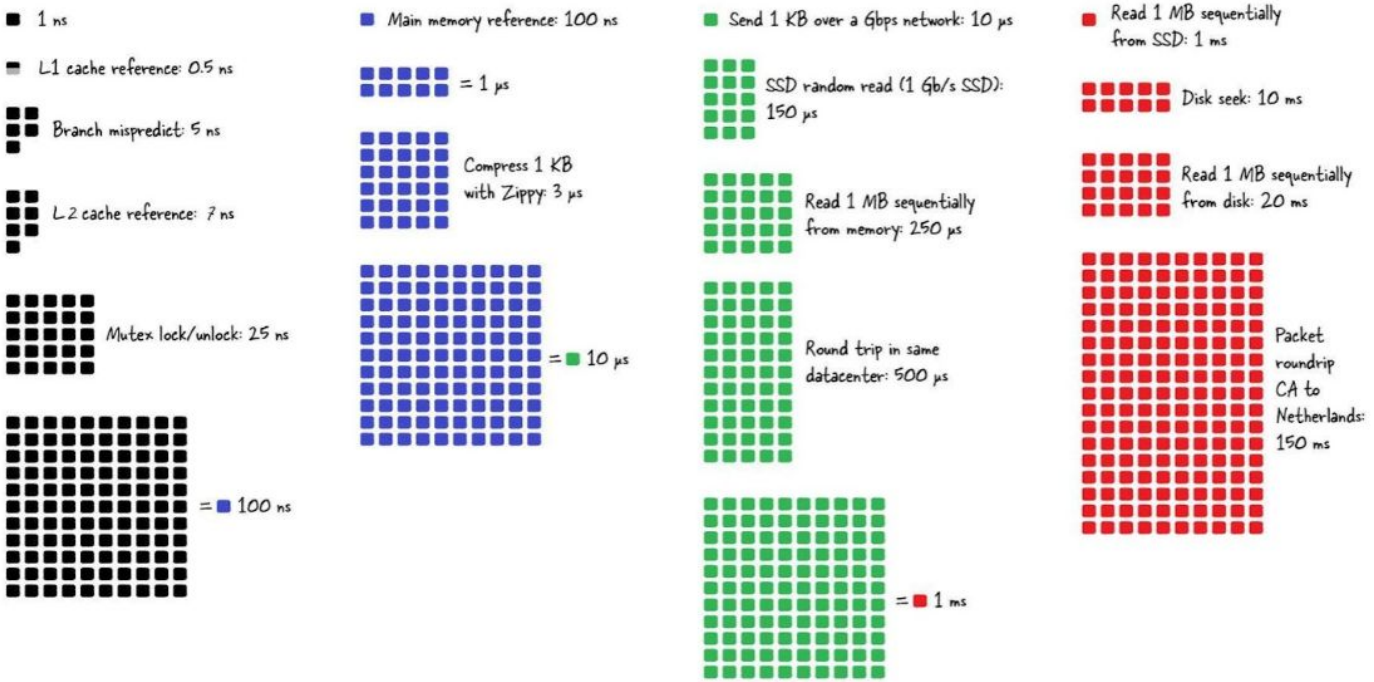
- репликация основной копии или ведущий-ведомый



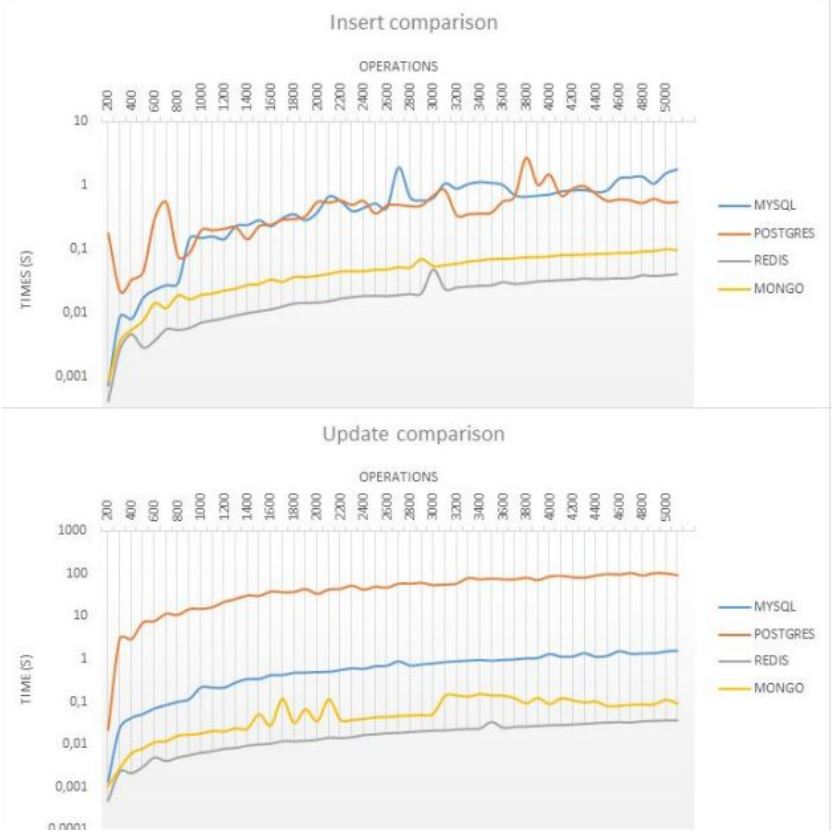

- одноранговая или симметричная репликация

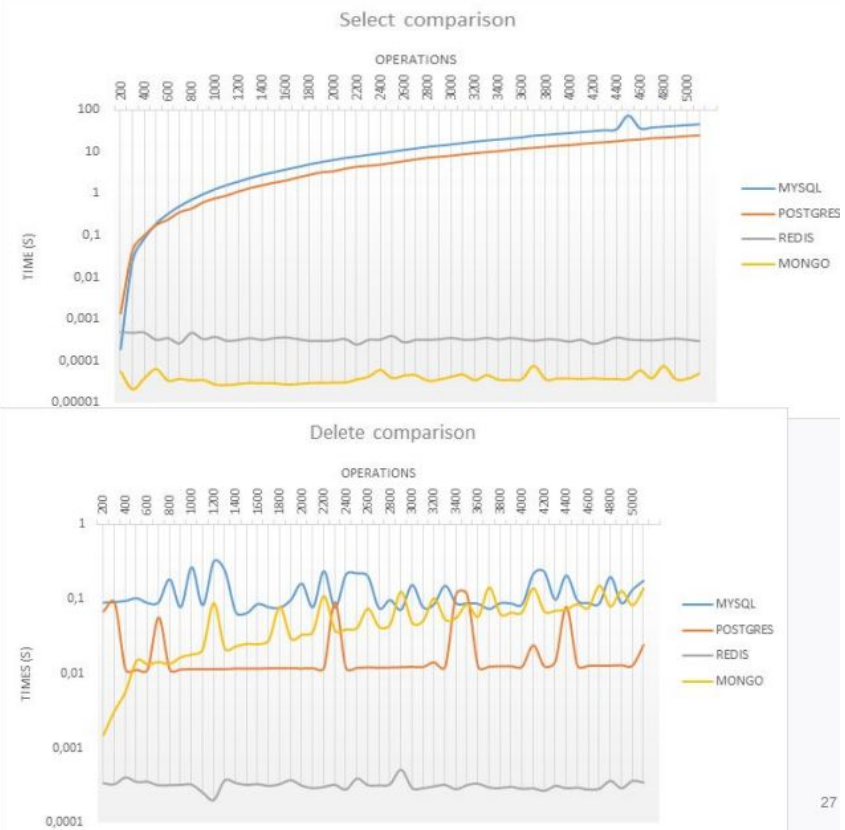


# Временные характеристики задержек и пропускной способности систем



# CRUD операции на SQL и NoSQL СУБД

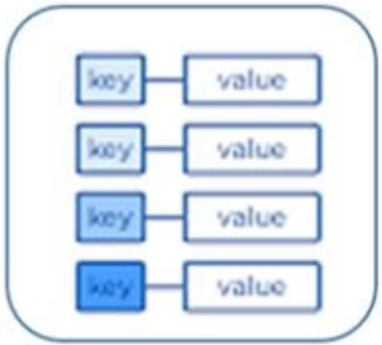


# Типы NoSQL СУБД

- Ключ - значение
- Документо-ориентированные
- Графовые
- Колоночные

Document Store




Key-Value Store

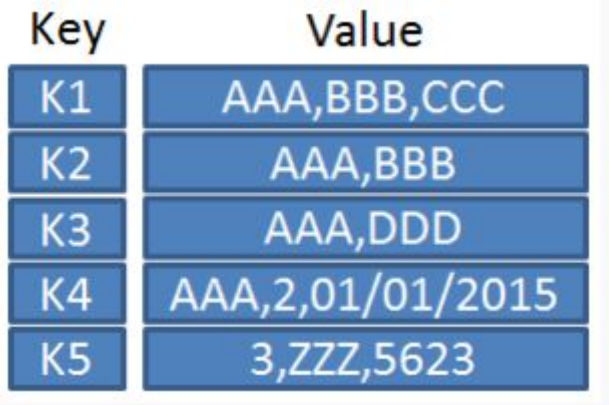
Wide-Column Store

Graph Store

# Типы NoSQL СУБД : Ключ - значение

Ключ-значение – под каждым ключом что-то лежит, пока не запросим – не узнаем что именно

- Redis
- DynamoDB
- Memcached



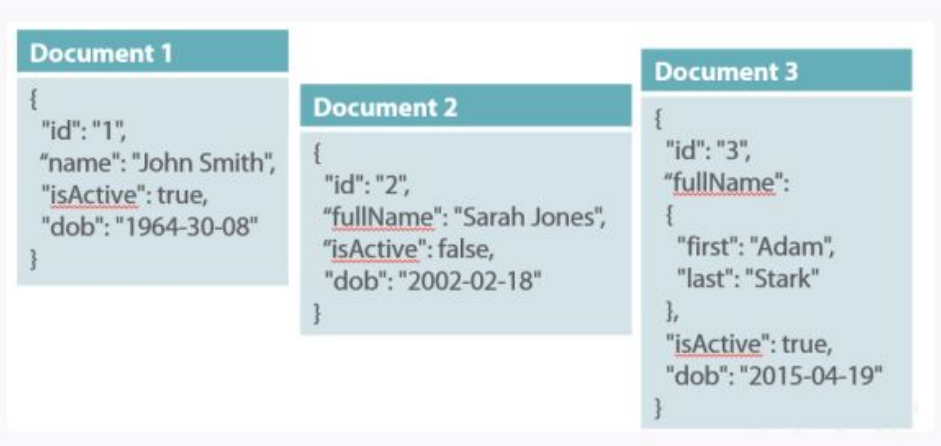
| Key | Value |
|---|---|
| K1 | AAA,BBB,CCC |
| K2 | AAA,BBB |
| K3 | AAA,DDD |
| K4 | AAA,2,01/01/2015 |
| K5 | 3,ZZZ,5623 |

# Типы NoSQL СУБД : Документно-ориентированная

Документно-ориентированная БД - аналог ключ-значение, но в качестве значений используется объекты в определенном формате (JSON, XML):

- Одиночные операции в CRUD выполняются гораздо быстрее
- Можно делать запросы к содержимому  записи, не извлекая данных целиком (сходство с RDBMS)

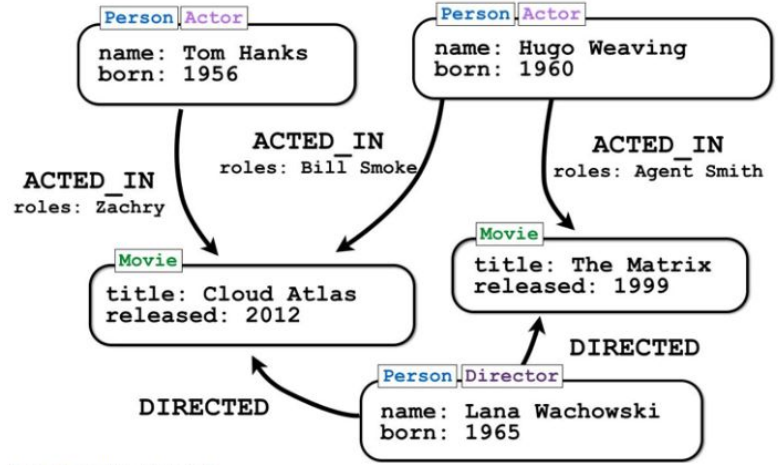
- MongoDB
- LiteDB
- CouchDB
- Elasticsearch



# Типы NoSQL СУБД : Графовые БД

Графовые БД – единицы хранения: Узлы и ребра(связи).
Запрос – обход данных от узла к узлу по ребрам на заданную глубину. Используются для хранения, управления, составления запросов к сложным тесно взаимосвязанным группам данных (социальные сети, сервисы рекомендаций, графы значений, логистика, геоинформационные системы.

- Neo4j
- OrientDb
- Titan

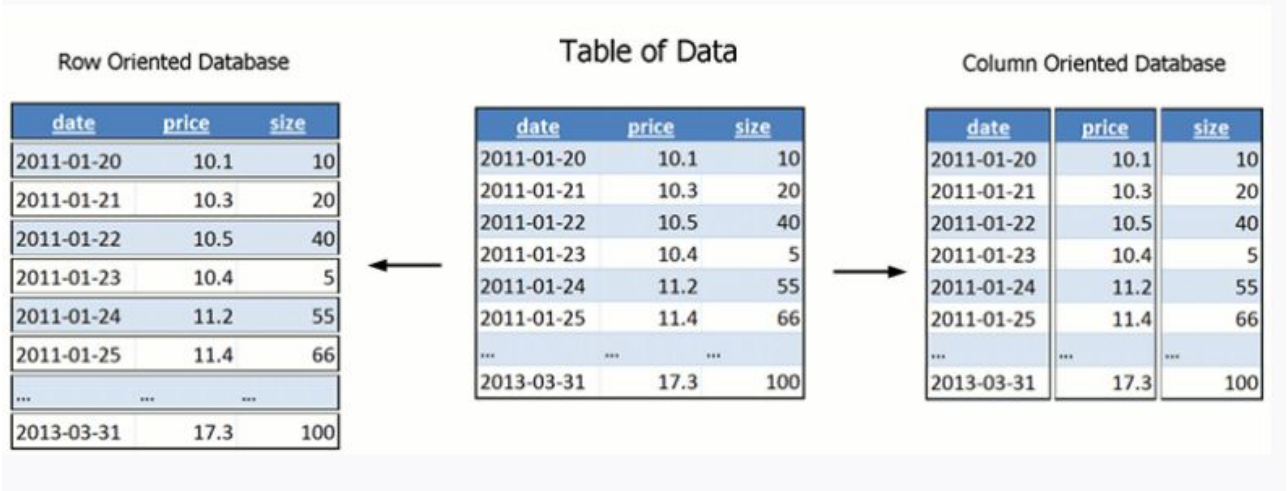



*A graph model with labels.*

# Типы NoSQL СУБД : Колоночные БД

Колоночные базы данных – данные хранятся в ячейках, сгруппированных в колонки
Применяются в веб-индексировании, рекламе, телекоммуникациях, аналитических система и т.п.

- Cassandra
- Hbase
- Google BigTable




| date | price | size |
|---|---|---|
| 2011-01-20 | 10.1 | 10 |
| 2011-01-21 | 10.3 | 20 |
| 2011-01-22 | 10.5 | 40 |
| 2011-01-23 | 10.4 | 5 |
| 2011-01-24 | 11.2 | 55 |
| 2011-01-25 | 11.4 | 66 |
| ... | ... | ... |
| 2013-03-31 | 17.3 | 100 |

# NoSQL СУБД : дополнительная информация


https://hostingdata.co.uk/nosql-database/

Количество NoSQL DB >225

https://db-engines.com/en/ranking

# NoSQL СУБД : как начать использовать

- Установить локально
- Поднять локально в контейнере (Docker)
- Использовать облачный сервис

# Redis

# Redis

- Ключ-значение
- Хранит данные в оперативной памяти, при необходимости может сохранять данные на диске
- Доступ по общему паролю или без него
- Без ключа данные не получить
- https://hub.docker.com/u/redis - офицальный образ Docker
- https://hub.docker.com/r/redis/redis-stack
- Хранилище структурированных данных
- Работа с различными типами данных
- Обычно используется как кэш но может использоваться как полноценное хранилище для небольших проектов или брокер очередей
- Отлично документированная СУБД
- Поддерживает широкий спектр языков программирования

# Redis

```
using StackExchange.Redis;
var redis = ConnectionMultiplexer.Connect("localhost");
var db = redis.GetDatabase();


db.StringSet(“MyKey", “MyValue");
string get_string = db.StringGet("MyKey");
```

HashSet:

```
db.HashSet("hashset", new [] {new HashEntry( “UserName", “Vasya" ) ,
       new HashEntry(“Age", “30") });
db.HashSet("hashset", “City", “Moscow");
var keys = db.HashKeys("hashset");
db.HashDelete("hashset","entry1");
var data = db.HashGetAll("hashset");
```

- Set

```
db.SetAdd("set_key", "value1");
db.SetAdd("set_key", "value2");
db.SetAdd("set_key", "value3");
var set_items = db.SetMembers("set_key");
var set_random_item = db.SetPop("set_key");
```

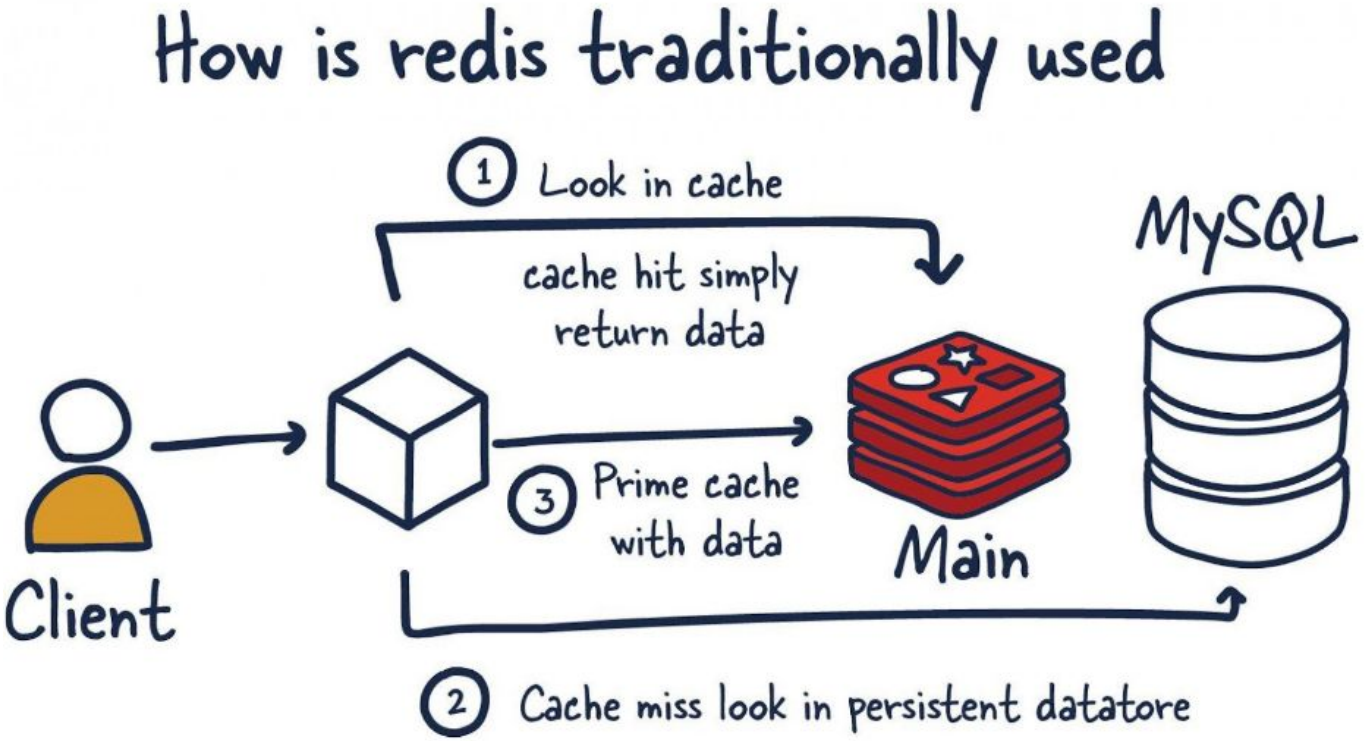
- List

```
db.ListLeftPush(“list", “item1");
db.ListRightPush("list", “item2");
var listdata = db.ListRange("u");
var item = db.ListRightPop("list");
```

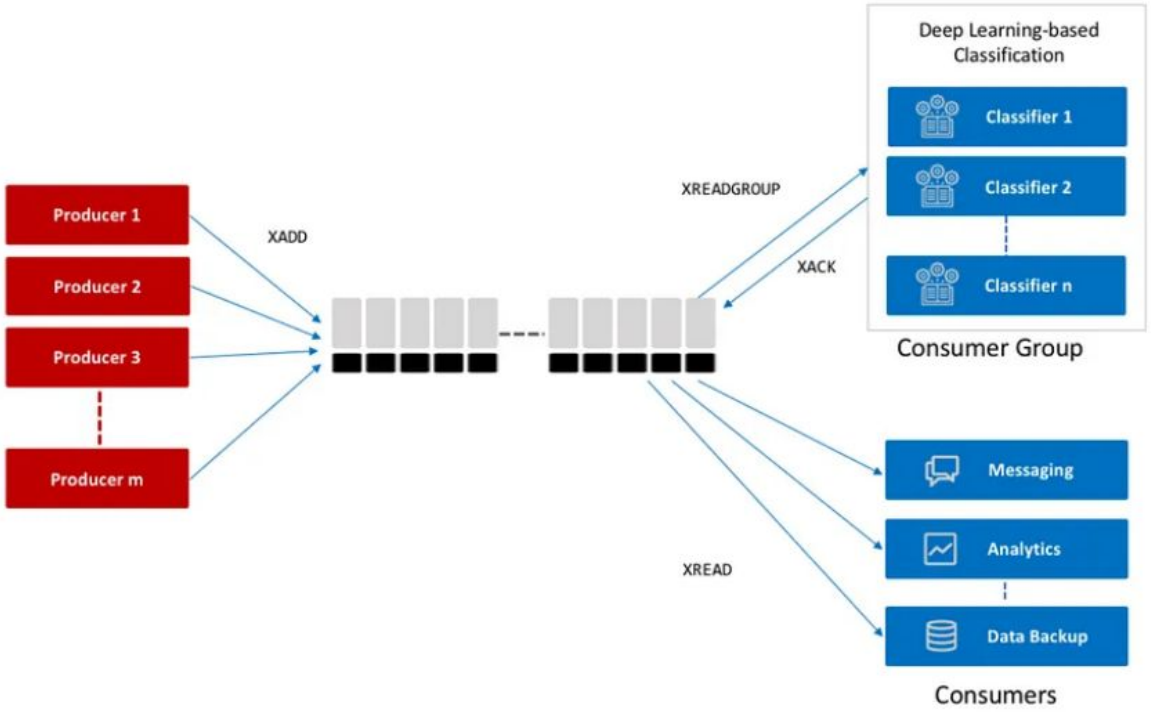
Транзакции:

```
var transaction = db.CreateTransaction();
transaction.HashSetAsync("hashset", new[] { new HashEntry("entry1", "4555"), new
            HashEntry("entry2", "4555") });
transaction.HashSetAsync("hashset", "entry3", "777");
transaction.HashDeleteAsync("hashset", "entry1");
transaction.Execute();
var data = db.HashGetAll("hashset");
```

# Redis -кэширование



# Redis - брокер очередей



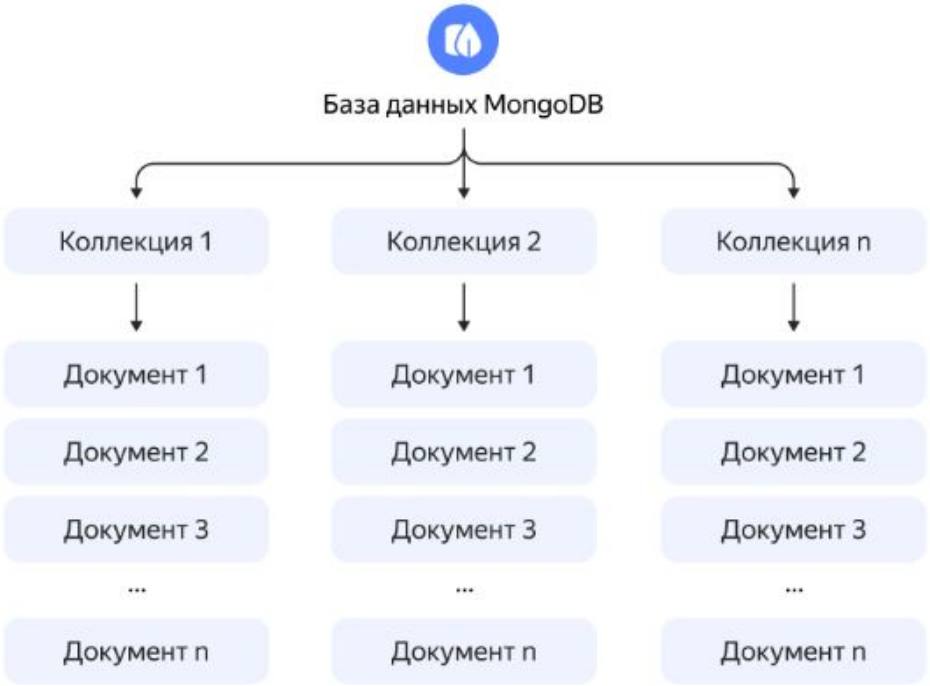
# LIVE

# MongoDB

# MongoDB

- Популярная документо-ориентированная БД
- Хранение данные в формате BSON
- Использование JavaScript для написания функций и запросов
- Возможность выполнения операций, аналогичных операциям в реляционных СУБД
- Поддержка хранимых процедур на JavaScript
- Валидация полей документов
- Возможность репликации
- Встроенная работа с гео-координатами
- Хорошо документированная СУБД
- Поддерживает широкий спектр языков программирования
- https://hub.docker.com/_/mongo
- https://www.mongodb.com/docs/drivers/csharp/current/

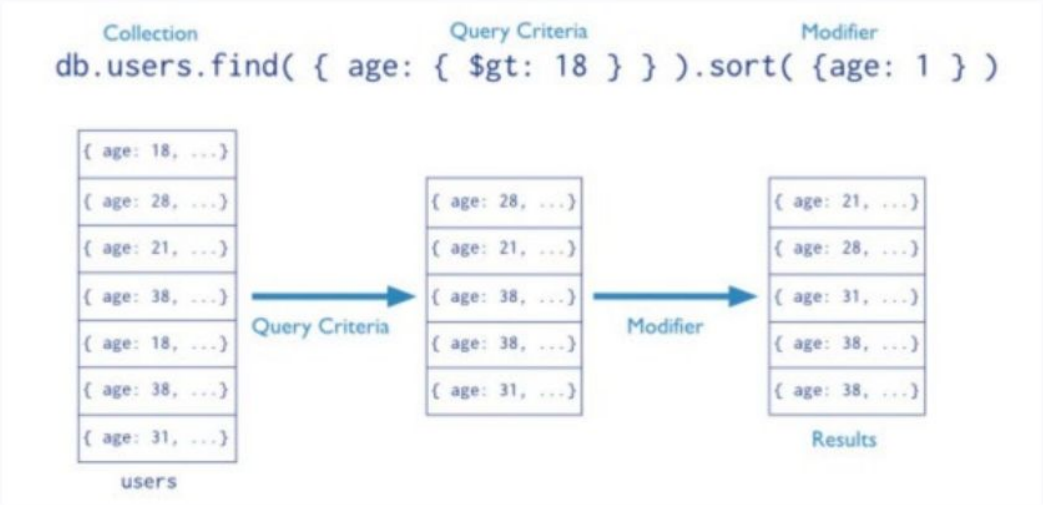
# MongoDB схема данных



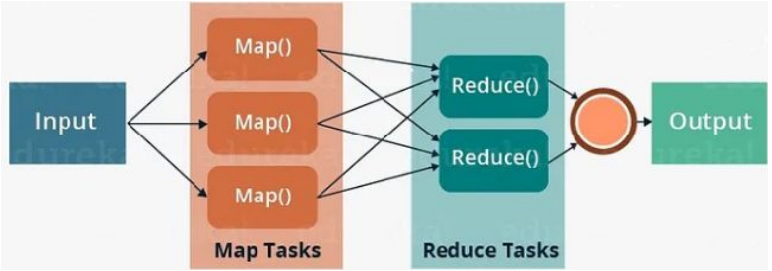
# MongoDB - операция выборка

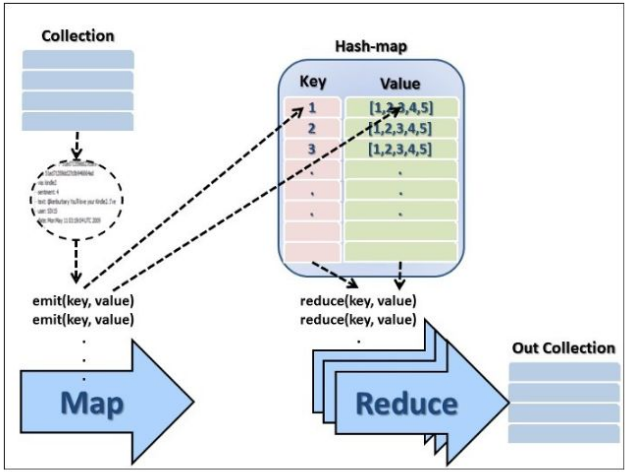
```
db.users.find(                          <- collection
   { age: { $gt: 18 } },                <- query criteria
   { name: 1, address: 1 }              <- projection
).limit(5)                              <- cursor modifier
```

```
SELECT _id, name, address   <- projection
FROM   users                <- table
WHERE  age > 18             <- select criteria
LIMIT  5                    <- cursor modifier
```



# MongoDB - паттерн Map Reduce - пайплайны аггрегации




```
db.train.aggregate( [
                    {$match:{class:"first-class"}},
                    {$group:{_id:"id",total:{$sum:"$fare"}}}
                   ])
```

pipeline stages

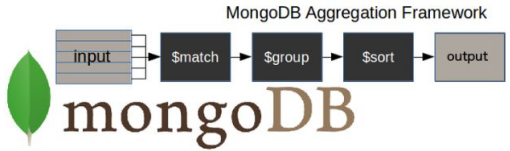
```
{
    id:"181",
    class:"first-class",
    fare: 1200
}
{
    id:"181",
    class:"first-class",
    fare: 1000
}
{
    id:"181",
    class:"second-class",
    fare: 1000
}
{
    id:"167",
    class:"first-class",
    fare: 1200
}
{
    id:"167",
    class:"second-class",
    fare: 1500
}
```

$match

```
{
    id:"181",
    class:"first-class",
    fare: 1200
}
{
    id:"181",
    class:"first-class",
    fare: 1000
}
{
    id:"167",
    class:"first-class",
    fare: 1200
}
```

$group

```
{
    _id:"181",
    total: 2200
}
{
    _id:"167",
    total: 1200
}
```




```
pipeline =
  [{$facet: {
      MyFacet1_OutputField: [
          <stage1>,
          <stage2>
      ],
      MyFacet2_OutputField: [
          <stage1>,
          <stage2>,
          <stage3>,
          <stage4>
      ]
  }}]
```

Sub-Pipeline 1

Sub-Pipeline 2

# MongoDB

Модель для работы

```
public class User
{
    public string _id { get; set; }
    public string user_name { get; set; }
    public int age { get; set; }
    public Company company { get; set; }
}

public class Company
{
    public string name { get; set; }
    public DateTime startwork { get; set; }
}
```

Вставка объекта

```
users.Insert(new User
        {
                age = 43,
                user_name = "Zaza",
                _id = Guid.NewGuid().ToString().ToLower(),
                company = new Company
                {
                        name = "Bueng",
                        startwork = DateTime.Now
                }
        });
```

Поиск по коллекции

```
var u = users.Find<User>(x => x.company.startwork >= new DateTime(2002, 1, 1)).ToList();
var p = users.FindOneAndDelete<User>(x => x.user_name == "Zaza");
```

Удаление элементов

```
users.DeleteMany<User>(x => x.user_name == "Zaza");
users.DeleteOne<User>(x => x.user_name == "Zaza");
```

Подключение к базе данных

```
var dbClient = new MongoClient("mongodb://127.0.0.1:27017");
IMongoDatabase db = dbClient.GetDatabase("Info");
var users = db.GetCollection<User>("Users");
```

Создание индексов

```
var index2 = Builders<User>.IndexKeys.Ascending(x => x.user_name);
users.Indexes.CreateOne(new CreateIndexModel<User>(index2));
```

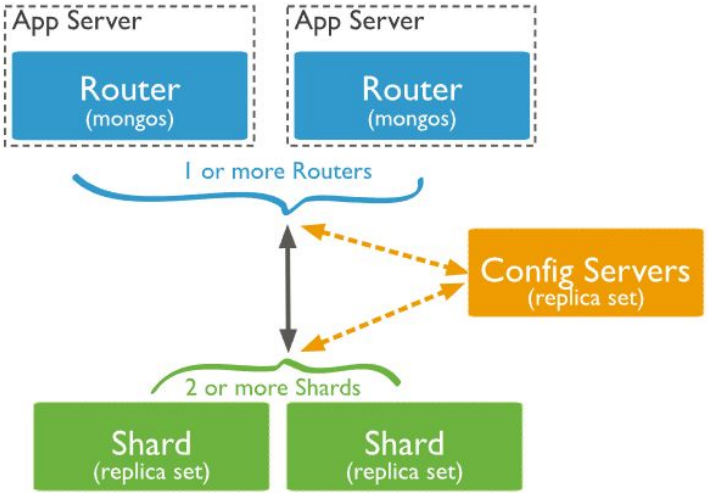
Обновление данных

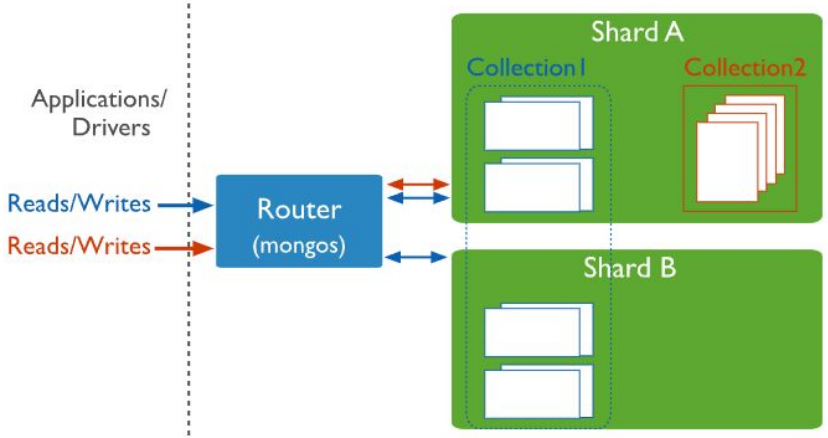
```
var update = Builders<User>.Update
                .Set(x => x.user_name, "Nana")
                .Set(x=>x.age, 25);

users.UpdateOne<User>(x => x.user_name == "Zaza", update);
users.UpdateMany<User>(x => x.company.name == "MegaHard", update);
```

Удаление элементов

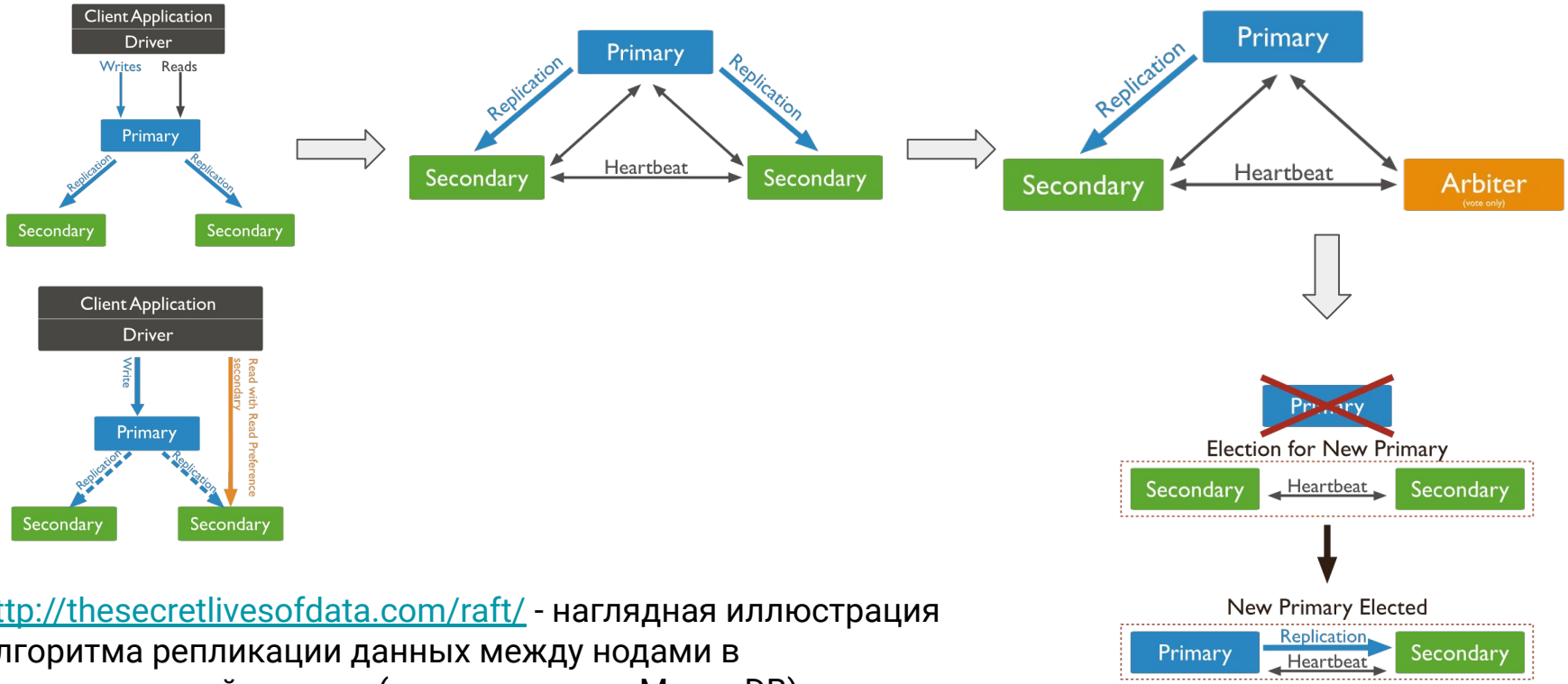
```
users.DeleteMany<User>(x => x.user_name == "Zaza");
users.DeleteOne<User>(x => x.user_name == "Zaza");
```

# MongoDB - шардирование



# MongoDB - репликация




http://thesecretlivesofdata.com/raft/ - наглядная иллюстрация алгоритма репликации данных между нодами в распределенной системе(используется в MongoDB)
https://www.mongodb.com/docs/manual/replication/

# LIVE

# LiteDB

# LiteDB

- Документо-ориентированная
- Standalone СУБД, не требует установки и конфигурирования
- Легковесная
- Позволяет быстро начать работать с документо-ориентированным хранилищем
- https://www.litedb.org/

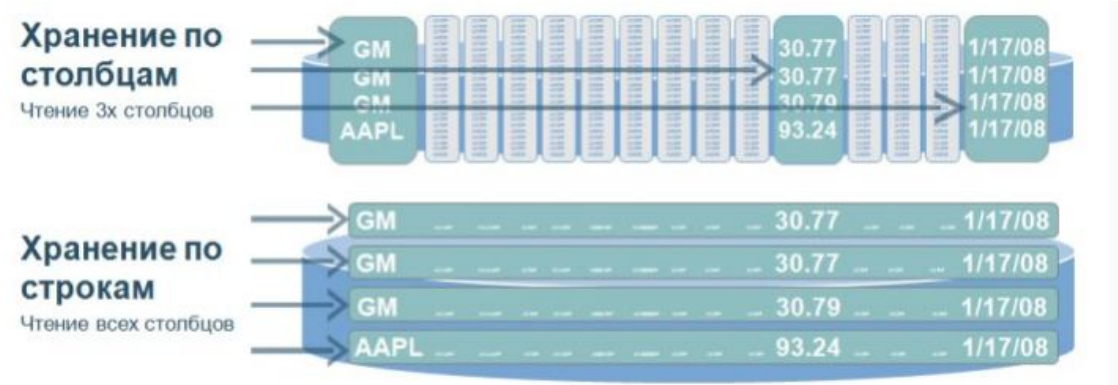
# Cassandra

# Cassandra

- Колонко-ориентированное хранилище
- Не реляционная отказоустойчивая распределенная СУБД
- Гибридное NoSQL решение , сочетает модель хранения на основе столбцов с моделью key-value
- Рассчитана на создание крупномасштабных надежных хранилищ, представленных в виде хеш
- Разработана на Java в 2008 году
- Наиболее удобная база данных для кластеризации
- Единственная база данных, где скорость записи выше скорости чтения (около 80-360 МБ/с на узел)

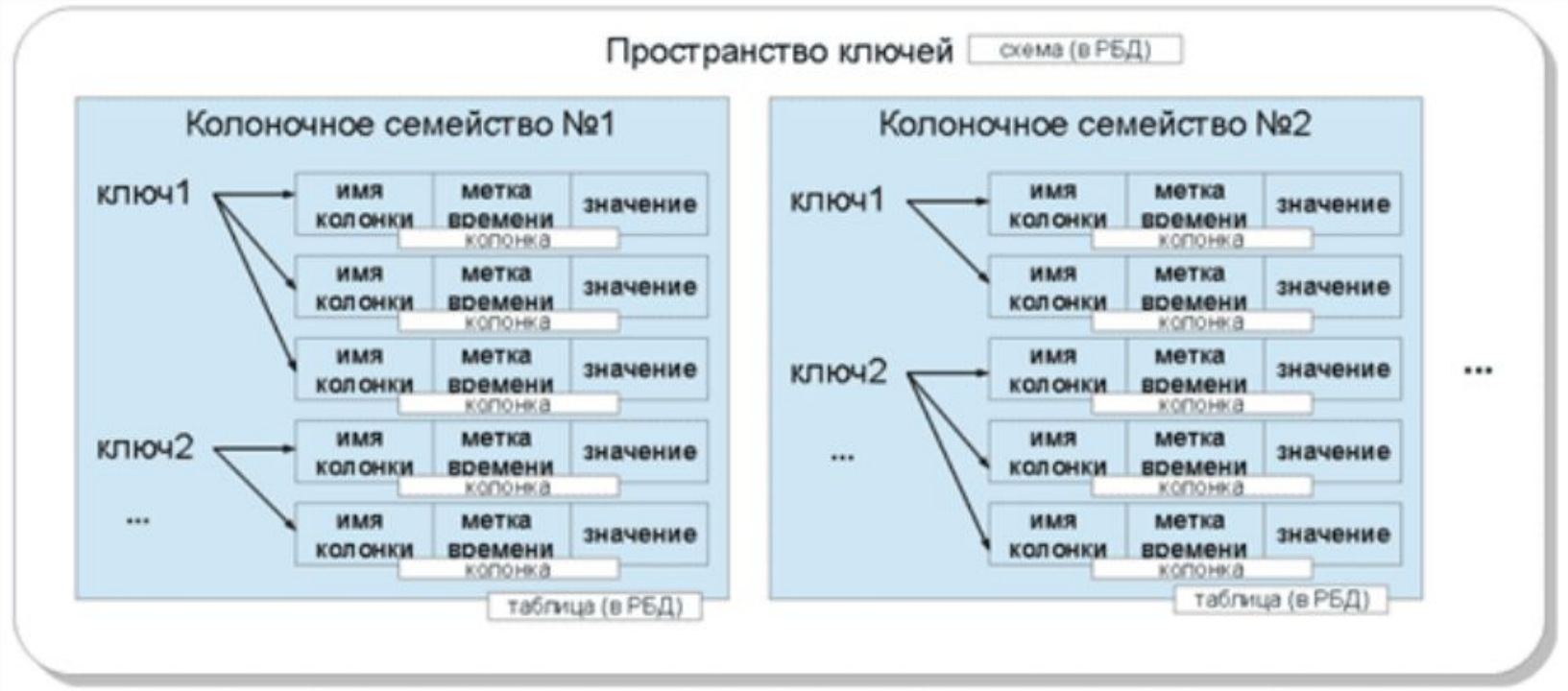
# Cassandra - модель данных

- **Столбец или колонка (column)** – ячейка с данными, включающая 3 части: имя (column name) в виде массива байтов, метку времени (timestamp) и само значение (value) также в виде байтового массива;
- **Строка или запись (row)** – именованная коллекция столбцов;
- **Семейство столбцов (column family)** – именованная коллекция строк;
- **Пространство ключей (keyspace)** – группа из нескольких семейств столбцов, собранных вместе.




Модель данных Apache Cassandra

# Cassandra - модель данных



# Cassandra - модель данных

- **Столбец или колонка (column)** – ячейка с данными, включающая 3 части: имя (column name) в виде массива байтов, метку времени (timestamp) и само значение (value) также в виде байтового массива;
- **Строка или запись (row)** – именованная коллекция столбцов;
- **Семейство столбцов (column family)** – именованная коллекция строк;
- **Пространство ключей (keyspace)** – группа из нескольких семейств столбцов, собранных вместе.

# Cassandra - создание пространства ключей

```
CREATE KEYSPACE [IF NOT EXISTS] keyspace_name
WITH REPLICATION = {
                'class' : 'SimpleStrategy', 'replication_factor' : N
                }
                | 'class' : 'NetworkTopologyStrategy', 'dc1_name' : N [, ...]
};
```

keyspace_name – название пространства ключей.
dc1_name – имя узла

**REPLICATION = { *replication_map* }**

Карта репликации определяет, сколько копий данных хранится в данном узле.
Этот параметр влияет на согласованность, доступность и скорость запроса.

Класс стратегии репликации и настройки факторов

| Класс | Фактор | Описание |
|---|---|---|
| 'SimpleStrategy' | 'replication_factor' : *N* | Для всего кластера будет использоваться один коэффициент репликации. Параметр N означает количество копий данных, должен быть целым числом. |
| 'NetworkTopologyStrategy' | '*datacenter_name*' : *N* | Для каждого узла задается свой коэффициент репликации. Параметр N означает количество копий данных, должен быть целым числом. |

Примеры задания топологий:
Простая топология:

```
'class' : 'SimpleStrategy', 'replication_factor' : 1
```

Сетевая топология:

```
'class' : 'NetworkTopologyStrategy', ‘London_dc’ : 1, ‘Moscow_dc’:2
```

# Cassandra

```
> CREATE TABLE IF NOT EXISTS test (
 id timeuuid,
 title text,
 PRIMARY KEY (title, id)
) WITH default_time_to_live = 120 and CLUSTERING ORDER BY (id DESC);
```

INSERT INTO student (id, citizenship, first_name, last_name, age) VALUES (now(), 'Russia', 'Ivan', 'Ivanov', 25);

Выполнив select –запрос отобразим все значения.

SELECT * FROM tablename;

select * from student;

```
cqlsh:lab7> select * from student;

 id                                   | age | citizenship | first_name | last_name
--------------------------------------+-----+-------------+------------+-----------
 3a26e100-9aeb-11ea-b1d1-3148925e06e7 |  25 |      Russia |       Ivan |    Ivanov
 6d0317a0-9aec-11ea-b1d1-3148925e06e7 |  22 |      France |       test |    Petrov
 47957270-9aeb-11ea-b1d1-3148925e06e7 |  35 |      Russia |       Petr |    Petrov
```

Пример UPDATE:
UPDATE имя_таблицы SET название_колонки=значение WHERE условие

Обновим поля first_name и last_name и строки с заданным id:

UPDATE student SET first_name = 'Example', last_name='Example'
WHERE id=54daf810-9aeb-11ea-b1d1-3148925e06e7

# Cassandra

Пример DELETE:
DELETE FROM название_таблицы WHERE условие.

DELETE FROM student WHERE id=54daf810-9aeb-11ea-b1d1-3148925e06e7:

СУБД не дает возможности по умолчанию искать по неключевым колонкам, для которых не создан индекс.

Запрос SELECT * FROM student WHERE citizenship='Russia' выдаст ошибку:

```
... ;
InvalidRequest: Error from server: code=2200 [Invalid query] message="Cannot execute this q
uery as it might involve data filtering and thus may have unpredictable performance. If you
 want to execute this query despite the performance unpredictability, use ALLOW FILTERING"
```

Решением этой проблемы является создание вторичного индекса или использование конструкции allow filtering.

Создадим индекс по колонке citizenship.
CREATE INDEX citizenshipIndex ON student (citizenship);
После этого можно выполнять запрос с условием на эту колонку.
SELECT * FROM student WHERE citizenship='Russia'

```
id                                   | age | citizenship | first_name | last_name
--------------------------------------+-----+-------------+------------+-----------
3a26e100-9aeb-11ea-b1d1-3148925e06e7 |  25 |      Russia |       Ivan |    Ivanov
47957270-9aeb-11ea-b1d1-3148925e06e7 |  35 |      Russia |       Petr |    Petrov
```

Если ваша таблица содержит, например, 1 миллион строк, и 95% из них имеют запрошенное значение для столбца citizenship,
запрос все равно будет относительно эффективным, и вам следует использовать ALLOW FILTERING.

Пример:
SELECT * FROM student WHERE last_name='Ivanov' ALLOW FILTERING;
В данном запросе будут отобраны все студенты с фамилией Иванов.

```
cqlsh:lab7> SELECT * FROM student WHERE last_name='Ivanov' ALLOW FILTERING;

 id                                   | age | citizenship | first_name | last_name
--------------------------------------+-----+-------------+------------+-----------
 3a26e100-9aeb-11ea-b1d1-3148925e06e7 |  25 |      Russia |       Ivan |    Ivanov
```

# Cassandra

Подключаемся к базе данных

```
Cluster cluster = Cluster.Builder().AddContactPoint("localhost").Build();
ISession session = cluster.Connect();
```

Готовим базу данных для использования

```
session.Execute("CREATE KEYSPACE uprofile WITH REPLICATION = { 'class' : 'NetworkTopologyStrategy', 'datacenter1' : 1 };");
session.Execute("CREATE TABLE IF NOT EXISTS uprofile.user (user_id int PRIMARY KEY, user_name text, user_bcity text)");
```

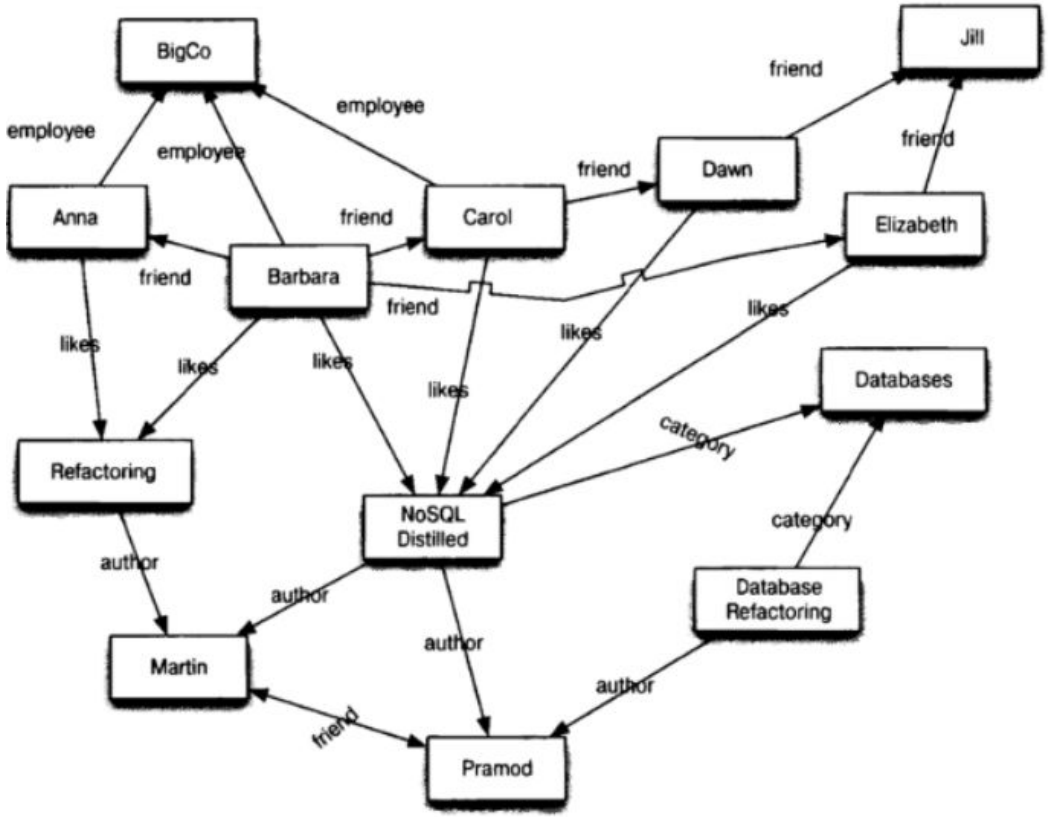
```
IMapper mapper = new Mapper(session);

mapper.Insert<User>(new User(1, "LyubovK", "Dubai"));
mapper.Insert<User>(new User(2, "JiriK", "Toronto"));
mapper.Insert<User>(new User(3, "IvanH", "Mumbai"));

IEnumerable<User> users = mapper.Fetch<User>("Select * from user");
var user = mapper.FirstOrDefault<User>("Select * from user where user_id = ?", 3);
```

```
public partial class User
   {
       public int user_id { get; set; }
       public string user_name { get; set; }
       public string user_bcity { get; set; }
   }
```

# Neo4j

# Neo4j

- Графовая база данных
- Свободное поле объектов
- Объекты соединяются определенными типами связей



# Neo4j

В качестве исходного материала

https://github.com/neo4j-examples/movies-dotnet-neo4jclient

```
docker run  --name testneo4j  -p7474:7474 -p7687:7687 -d -v c:/neo4j/data:/data -v c:/neo4j/logs:/logs  -v
c:/neo4j/import:/var/lib/neo4j/import  -v c:/neo4j/plugins:/plugins --env NEO4J_AUTH=neo4j/test  neo4j:latest
```

Подключаемся к базе данных

```
var client = new GraphClient(new Uri("http://localhost:7474"), "neo4j", "test");
client.ConnectAsync().Wait();
```

Получаем данные

```
var query = client.Cypher.
Match("(m:Movie)<-[:ACTED_IN]-(a:Person)")
.Return((m, a) => new
{
        movie = m.As<Movie>().title,
        cast = Return.As<string>("collect(a.name)")
}).Limit(100);
var data = query.ResultsAsync.Result.ToList();
```

# Список материалов для изучения


1. https://en.wikipedia.org/wiki/Database
2. https://en.wikipedia.org/wiki/Internet
3. https://en.wikipedia.org/wiki/Strozzi_NoSQL
4. https://www.geeksforgeeks.org/difference-between-sql-and-nosql/?ref=ml_lbp
5. https://phoenixnap.com/kb/acid-vs-base
6. https://ru.wikipedia.org/wiki/ACID
7. https://habr.com/ru/articles/555920/
8. https://www.yuji.page/acid/
9. https://cloud.yandex.ru/blog/posts/2022/10/nosql
10. https://ru.wikipedia.org/wiki/%D0%A2%D0%B5%D0%BE%D1%80%D0%B5%D0%BC%D0%B0_CAP
11. https://www.infoq.com/articles/cap-twelve-years-later-how-the-rules-have-changed/
12. https://habr.com/ru/articles/328792/
13. https://ru.wikipedia.org/wiki/%D0%A2%D0%B5%D0%BE%D1%80%D0%B5%D0%BC%D0%B0_PACELC
14. https://bigdataschool.ru/blog/cap-and-pacelc-in-kafka.html
15. https://web-creator.ru/articles/partitioning_replication_sharding
16. https://dzen.ru/a/ZElAFXc8MhP8ZU5K
17. https://martinfowler.com/books/nosql.html
18. https://www.diva-portal.org/smash/get/diva2:1681550/FULLTEXT01.pdf
19. https://learn.microsoft.com/ru-ru/dotnet/architecture/cloud-native/relational-vs-nosql-data
20. https://highload.guide/blog/NoSQL-quick-facts.html
21. https://hostingdata.co.uk/nosql-database/
22. https://db-engines.com/en/ranking
23. https://coderlessons.com/articles/java/osnovnye-kliuchi-mongodb-vash-drug
24. https://www.mongodb.com/docs/manual/tutorial/getting-started/
25. https://www.mongodb.com/docs/manual/sharding/
26. https://www.mongodb.com/docs/manual/replication/
27. http://thesecretlivesofdata.com/raft
28. https://www.postgresql.eu/events/pgconfeu2017/sessions/session/1596/slides/29/Distributed%20Computing%20on%20PostgreSQL.pdf
29. https://habr.com/ru/companies/piter/articles/275633/
30. https://xebia.com/blog/microservices-coupling-vs-autonomy/

# Список материалов для изучения


31. https://redis.io/
32. https://hub.docker.com/u/redis
33. https://habr.com/ru/companies/wunderfund/articles/685894/
34. https://www.mongodb.com/
35. https://www.litedb.org/
36. https://cassandra.apache.org/_/quickstart.html
37. https://www.cockroachlabs.com/
38. https://habr.com/ru/companies/flant/articles/327640/
39. https://devathon.com/blog/cockroachdb-vs-mysql-vs-postgresql-vs-mongodb-vs-cassandra/
40. https://blog.yakunin.dev/cockroachdb-postgresql/
41. https://www.digitalocean.com/community/tutorials/understanding-database-sharding
42. https://www.cockroachlabs.com/blog/limits-of-the-cap-theorem/
43. https://jepsen.io/consistency
44. http://www.cs.umd.edu/~abadi/papers/abadi-pacelc.pdf
45. https://redis.io/docs/latest/develop/data-types/streams/
46. https://github.com/neelabalan/mongodb-sample-dataset/tree/main
47. https://www.geeksforgeeks.org/datatypes-in-mongodb/
48. https://www.mongodb.com/docs/manual/reference/bson-types/
49. https://www.mongodb.com/docs/manual/core/views/create-view/

# Вопросы?

Ставим “+”,
если вопросы есть

Ставим “–”,
если вопросов нет

# Рефлексия

# Заполните, пожалуйста, опрос о занятии по ссылке в чате

Спасибо за внимание!

# Приходите на следующие вебинары

**Михаил Дмитриев**

**Ведущий программист НИПК Электрон**

Разрабатываю и поддерживаю приложения для работы с радиологическими комплексами

https://t.me/sf321