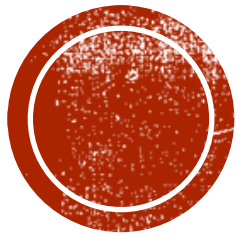
ЯЗЫКИ ПРИКЛАДНОГО ПРОГРАММИРОВАНИЯ

Лекция 6

```
public static class StringExtension
{
    public static int CharactersCount(this string str, char c)
    {
        var counter = 0;
        for (var i = 0; i < str.Length; i++)
        {
            if (str[i] == c)
                counter++;
        }
        return counter;
    }
}
```

МЕТОДЫ РАСШИРЕНИЯ

- Методы расширения (extension methods) позволяют добавлять новые методы в уже существующие типы без создания нового производного класса. Эта функциональность бывает особенно полезна, когда нам хочется добавить в некоторый тип новый метод, но сам тип (класс или структуру) мы изменить не можем, поскольку у нас нет доступа к исходному коду.



LINQ

LINQ (LANGUAGE-INTEGRATED QUERY)

- язык запросов к источнику данных.
- Варианты источников данных:
  - IEnumerable (**LINQ to Objects**),
  - DataSet (**LINQ to DataSet, ADO .NET**),
  - База данных (**LINQ to Entities, EF**),
  - документ XML …
  - …
- Вне зависимости от типа источника LINQ позволяет применить ко всем один и тот же подход.

```
public static void UsingOperators()
{
    string[] people = { "Tom", "Bob", "Sam", "Tim", "Tomas", "Bill" };

    // создаем новый список для результатов
    var selectedPeople = from p in people
        // передаем каждый элемент из people в переменную p
        where p.ToUpper().StartsWith("T") //фильтрация по критерию
        orderby p // упорядочиваем по возрастанию
        select p; // выбираем объект в создаваемую коллекцию

    foreach (var person in selectedPeople)
        Console.WriteLine(person);
}

public static void UsingExtensions()
{
    string[] people = { "Tom", "Bob", "Sam", "Tim", "Tomas", "Bill" };

    var selectedPeople = people
        .Where(p => p.ToUpper().StartsWith("T"))
        .OrderBy(p => p);

    foreach (var person in selectedPeople)
        Console.WriteLine(person);
}
```

СПОСОБ ПРИМЕНЕНИЯ

- Операторы запросов
- Методы расширений

ГОТОВИМСЯ

```
public class UserAction
{
    public UserAction(string actionText, DateTime date)
    {
        ActionText = actionText;
        Date = date;
    }

    public string ActionText { get; }

    public DateTime Date { get; }
}

string[] people = { "Tom", "Bob", "Sam", "Tim", "Tomas", "Bill" };
```

WHERE

- Фильтрация значений коллекции по условию

```
string[] people = { "Tom", "Bob", "Sam", "Tim", "Tomas", "Bill" };

var where = people.Where(x => x.StartsWith("T"));
```

SELECT

- Проекция коллекции

```
var hellos = people.Select(x => $"Hello, {x}!");
var lengths = people.Select(x => x.Length);
```

SELECT MANY

- Проекция коллекции, объединяющая коллекции

```
int[][] arrays = {
    new[] {1, 2, 3},
    new[] {4},
    new[] {5, 6, 7, 8},
    new[] {12, 14}
};

// Will return { 1, 2, 3, 4, 5, 6, 7, 8, 12, 14 }
var result = arrays.SelectMany(array => array);
```

ANY & ALL

- **All**: определяет, все ли элементы коллекции удовлетворяют определенному условию
- **Any**: определяет, удовлетворяет хотя бы один элемент коллекции определенному условию

```
var haveSam = people.Any(x => x == "Sam");

var haveNoEmptyStrings = people.All(x => !string.IsNullOrWhiteSpace(x));
```

COUNT

- Подсчитывает количество элементов коллекции, которые удовлетворяют определенному условию

```
var peoplesCount = people.Count();

var peoplesStartsFromTCount = people.Count(x => x.StartsWith("T"));
```

```
UserAction[] actions = new[]
{
    new UserAction("Authorize", DateTime.Now.AddDays(-1)),
    new UserAction("Authorize", DateTime.Now.AddDays(-2)),
    new UserAction("Authorize", DateTime.Now.AddDays(-3)),

    new UserAction("Update name", DateTime.Now.AddDays(-2).AddHours(-2)),
    new UserAction("Update email", DateTime.Now.AddDays(-2)),
    new UserAction("Confirm email", DateTime.Now.AddDays(-2)),
    new UserAction("Fill balance", DateTime.Now.AddDays(-1)),
};

var ordered = actions.OrderBy(x => x.Date);

var orderedByDesc = actions.OrderByDescending(x => x.Date);
```

ORDER BY & ORDER BY DESCENDING

- **OrderBy**: упорядочивает элементы по возрастанию
- **OrderByDescending**: упорядочивает элементы по убыванию

FIRST & FIRST OR DEFAULT

- **First**: выбирает первый элемент коллекции (опционально – по условию)
- **FirstOrDefault**: выбирает первый элемент коллекции (опционально – по условию) или возвращает значение по умолчанию

```
var noActions = new List<UserAction>();

var firstName = people.First();
var firstNameStartsFromT = people.First(x => x.StartsWith("T"));
var firstUpdateAction = actions.First();
actions.First(); // exception

var defaultAction = actions.FirstOrDefault(); // null
```

LAST & LAST OR DEFAULT

- Как **First**, только **Last** =)

SINGLE & SINGLE OR DEFAULT

- **Single**: выбирает единственный элемент коллекции, если коллекция содержит больше или меньше одного элемента, то генерируется исключение
- **SingleOrDefault**: выбирает единственный элемент коллекции. Если коллекция пуста, возвращает значение по умолчанию. Если в коллекции больше одного элемента, генерирует исключение

```
var singleConfirm = actions.Single(x => x.ActionText.Contains("Confirm"));
var singleAuthorize = actions.Single(x => x.ActionText == "Authorize"); // exception
var single = actions.Single(); // exception

defaultAction = actions.SingleOrDefault(); // exception
defaultAction = actions.SingleOrDefault(x => string.IsNullOrWhiteSpace(x.ActionText)); null
```

SUM, AVERAGE, MIN, MAX

```
var numbers = new List<int>() { 1, 3, 2, 1};

var numbersSum = numbers.Sum(); // 7
var min = numbers.Min(); // 1
var max = numbers.Max(); // 3
var avg = numbers.Average(); // 7 / 4
```

AGGREGATE

- **Aggregate**: применяет к элементам последовательности агрегатную функцию, которая сводит их к одному объекту

```
var names = people.Aggregate("Names:", (first, next) => $"{first} {next}");
// Names: Tom, Bob, Sam, Tim, Tomas, Bill
```

DISTINCT

- **Distinct**: удаляет дублирующийся элементы из коллекции

```
var distinctNumbers = numbers.Distinct(); // 1, 2, 3
```

TAKE & SKIP

- **Take**: выбирает определенное количество элементов
- **Skip**: пропускает определенное количество элементов

```
var firstTwoActions = actions.Take(2);

var skipTwoActions = actions.Skip(2);

var takeTwoActionsInSecondPage = actions.Skip(2).Take(2);
```

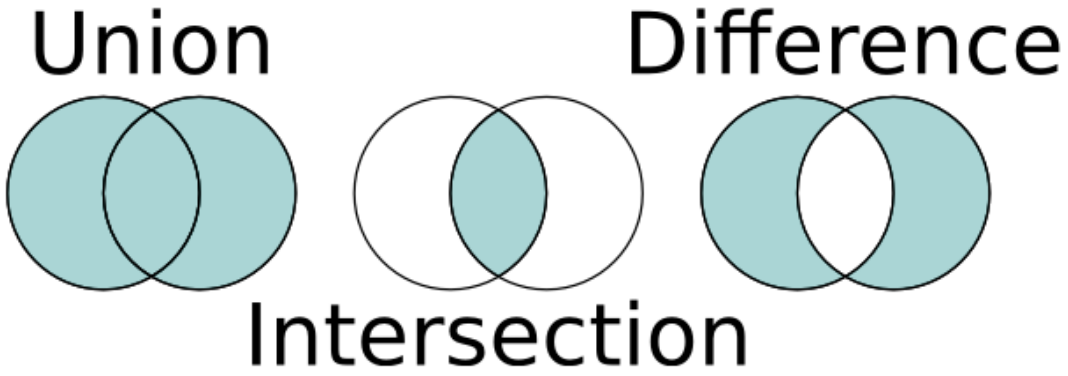
GROUP BY & TOLOOKUP

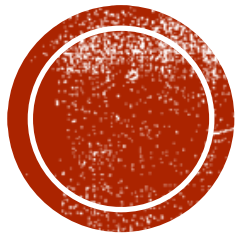
- **GroupBy**: группирует элементы по ключу
- **ToLookup**: группирует элементы по ключу, при этом все элементы добавляются в словарь

```
var actionsByDate = actions.GroupBy(x => x.Date);

var actionsByDateDictionary = actions.ToLookup(x => x.Date);
```

EXCEPT, UNION, INTERSECT

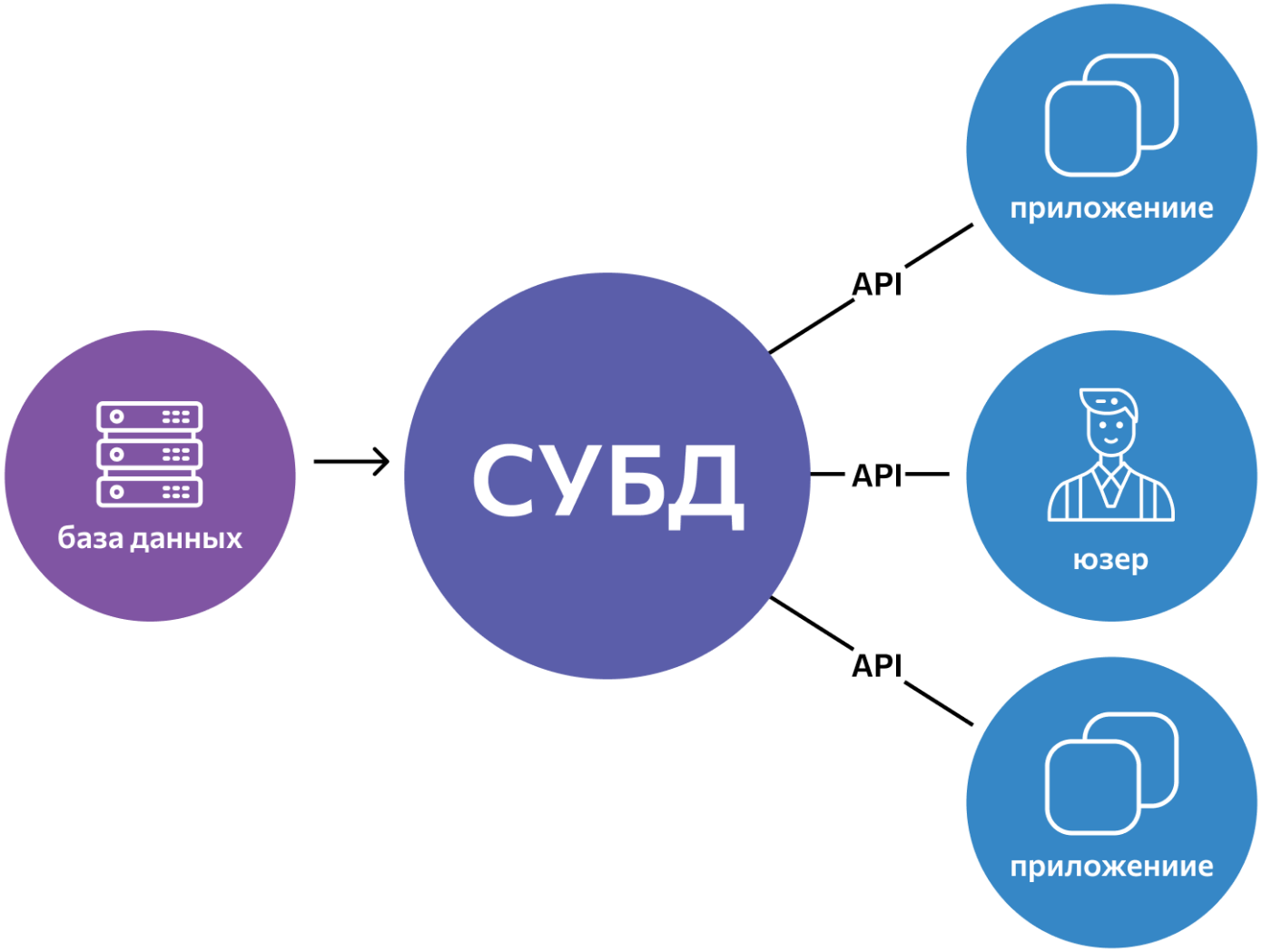




БАЗЫ ДАННЫХ

ЧТО ТАКОЕ БАЗА ДАННЫХ?

- База данных (БД) это:
  - имеющая название совокупность данных, которая отражает состояние объектов и их отношений в рассматриваемой предметной области.
  - Упорядоченный набор структурированной информации или данных, которые хранятся в электронном виде в компьютерной системе.
  - …
- Определений много, но важно понимать, что БД:
  - Это структурированная совокупность данных
  - Хранится в информационной системе (зачастую в долговременной памяти)
- Полезная статья на хабре



ЧТО ТАКОЕ СУБД?

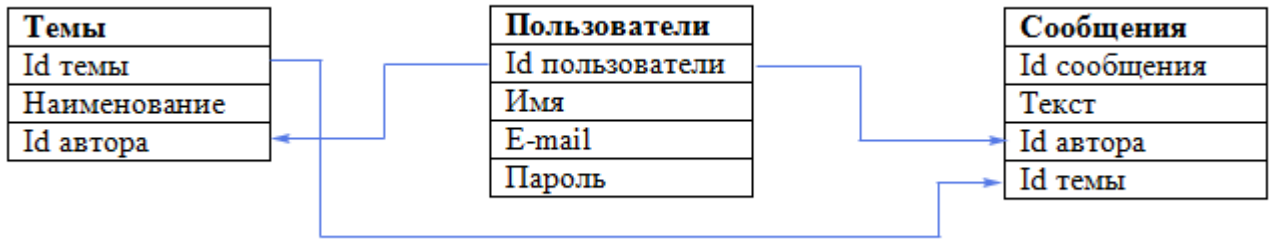
- СУБД (система управления базами данных) – это ПО, предназначенное для работы с базами данных.
- СУБД:
  - Это интерфейс между БД и пользователем/ПО использующим эту БД.
  - Предоставляет возможность получать, обновлять информацию, а также управлять ее упорядочением и оптимизацией.
  - Обеспечивает контроль и управление данными (администрирование, контроль, мониторинг, настройка, восстановление и т.п).

ТИПЫ БАЗ ДАННЫХ

- Реляционные (**SQL**)
- Не реляционные (**NoSQL**)
  - Документные
  - Графовые
  - Колоночные
  - Ключ-значение
  - …

РЕЛЯЦИОННЫЕ БД

- Данные организуются в виде набора таблиц, состоящих из столбцов и строк.
- Каждый столбец имеет строго определенный тип данных.
- Каждая стока таблицы - набор связанных значений, относящихся к одному объекту или сущности.
- Ячейка – значение атрибута сущности.

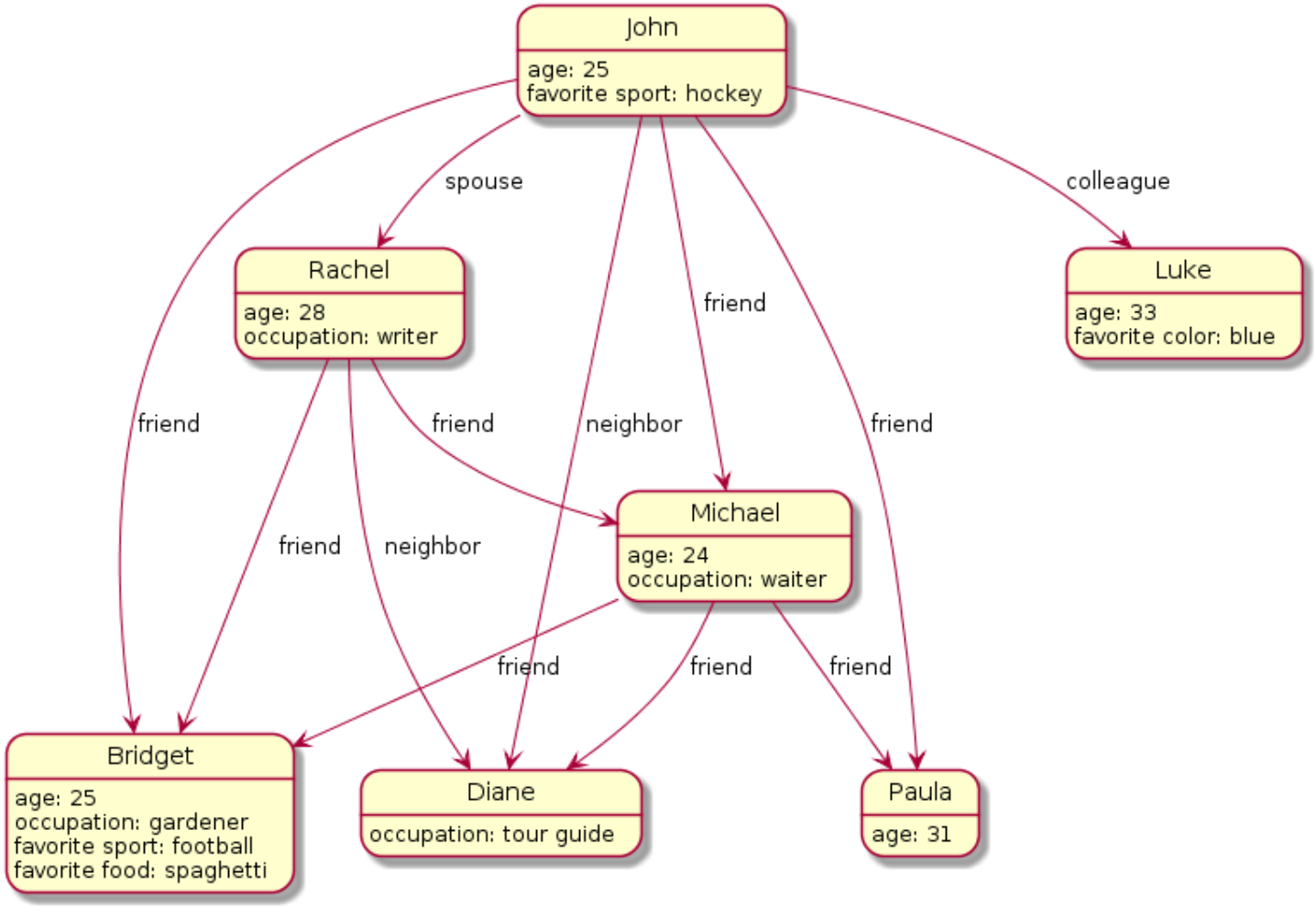


```json
{
  "address": {
     "building": "1007",
     "coord": [ -73.856077, 40.848447 ],
     "street": "Morris Park Ave",
     "zipcode": "10462"
  },
  "borough": "Bronx",
  "cuisine": "Bakery",
  "grades": [
     { "date": { "$date": 1393804800000 }, "grade": "A", "score": 2 },
     { "date": { "$date": 1378857600000 }, "grade": "A", "score": 6 },
     { "date": { "$date": 1358985600000 }, "grade": "A", "score": 10 },
     { "date": { "$date": 1322006400000 }, "grade": "A", "score": 9 },
     { "date": { "$date": 1299715200000 }, "grade": "B", "score": 14 }
  ],
  "name": "Morris Park Bake Shop",
  "restaurant_id": "30075445"
}
```

ДОКУМЕНТНЫЕ/ОБЪЕКТНЫЕ БД

- Хранятся коллекции документов/объектов
- В общем случае в одной коллекции могут быть разные типы/форматы объектов
- Хранят данные в форматах **JSON (BSON)**, **XML** и т.п.

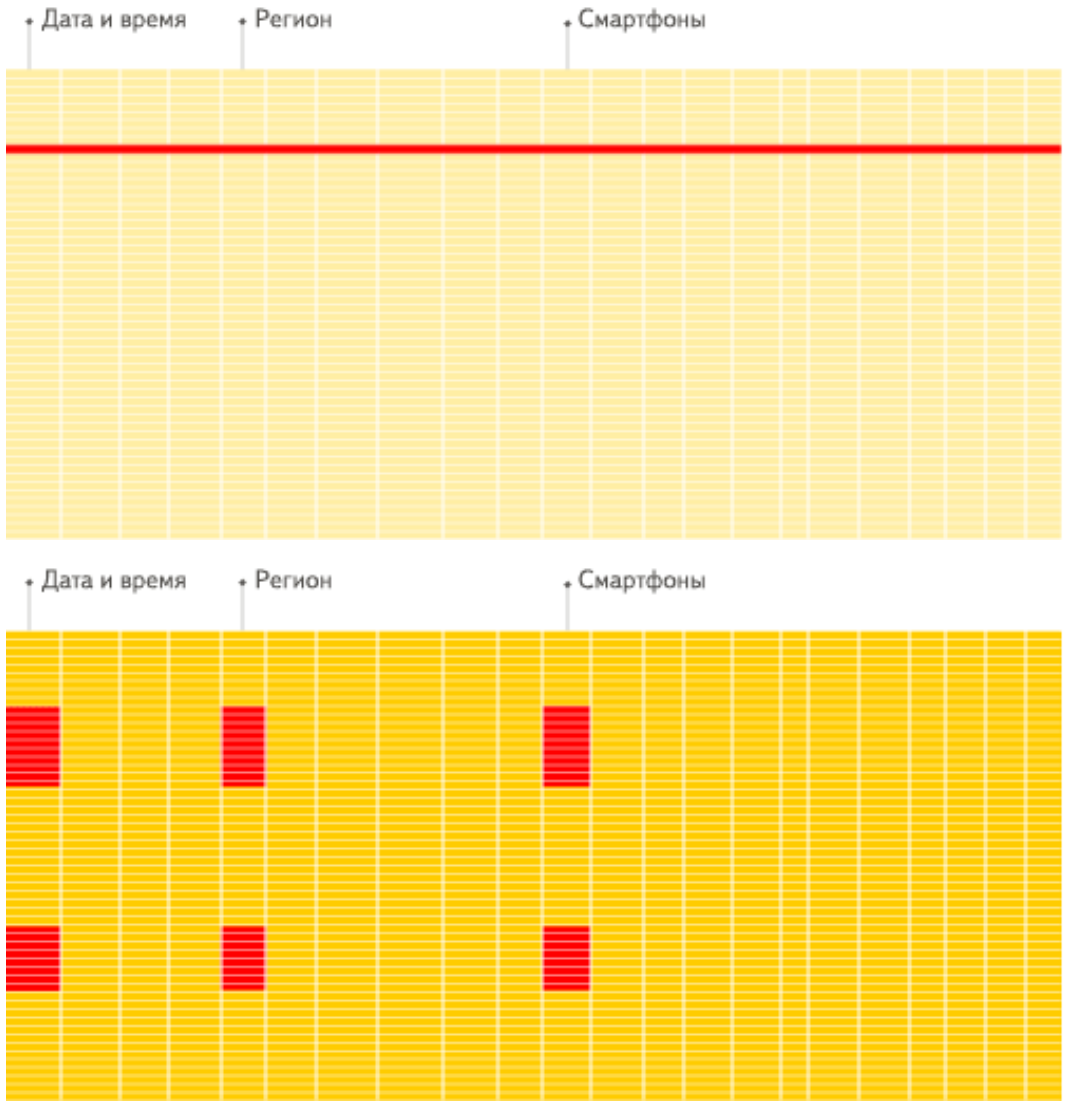
- Характерные представители:
  - **MongoDB**
  - **Amazon DocumentDB**
  - **RethinkDB**



ГРАФОВЫЕ БД

- Данные хранятся в виде графа
- Фокус на отношениях между сущностями

- Примеры:
  - **Neo4j**
  - **JanusGraph**
  - **Dgraph**



КОЛОНОЧНЫЕ БД

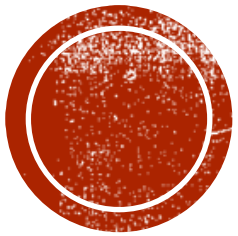
- Данные хранятся по колонкам, а не строкам.
- Дешевое чтение, добавление компактное хранение
- Дорогое удаление и изменение

- Примеры:
  - **ClickHouse**
  - **Cassandra**

БД ТИПА КЛЮЧ-ЗНАЧЕНИЕ

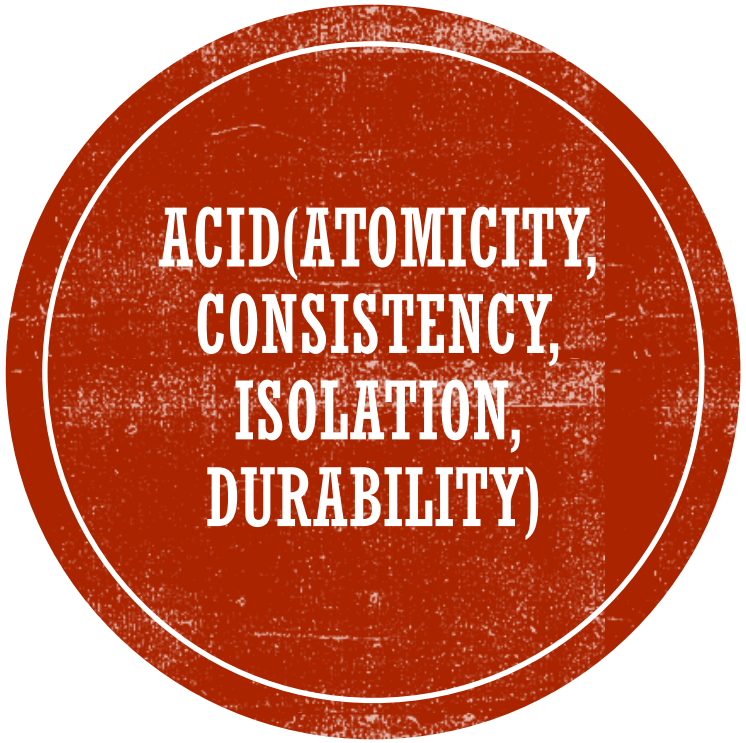
- Совокупность пар ключ-значение (словарь/**hash-map**).
- Как ключи, так и значения могут представлять собой что угодно: от простых до сложных составных объектов.

- Примеры:
  - **Redis**
  - **DynamoDB**



РЕЛЯЦИОННЫЕ БД

Чуть подробнее…



- **ACID** - набор требований к системе, обеспечивающий наиболее надёжную и предсказуемую её работу:
  - атомарность,
  - согласованность,
  - изоляция,
  - надежность;
- Требования сформулированы в конце 1970-х годов.
- Реляционные СУБД соответствуют требованиям **ACID**, не реляционные, зачастую – нет.
- **https://habr.com/ru/post/555920/**

АТОМАРНОСТЬ

- Транзакция – это одно или несколько действий, выполненных в виде последовательности операций, представляющих собой единую логическую задачу.
- **Атомарность** – это условие, при котором либо транзакция успешно выполняется целиком, либо, если какая-либо из ее частей не выполняется, вся транзакция отменяется.
- Т.е. транзакция – неделимое (атомарное) действие.

СОГЛАСОВАННОСТЬ

- **Согласованность** – это условие, при котором данные, записываемые в базу данных в рамках транзакции, должны соответствовать всем правилам и ограничениям, включая ограничения целостности, каскады и триггеры.
- Поддерживается за счет:
  - Первичных ключей,
  - Внешних ключей,
  - Ограничений на значения столбцов
- Ограничения позволяют применять правила предметной области к данным в таблицах и гарантировать точность и надежность данных.
- Большинство ядер БД также поддерживает выполнение SQL-скрипта, который выполняется в ответ на определенные операции в БД (триггеры).

ПЕРВИЧНЫЙ И ВНЕШНИЙ КЛЮЧИ

- Каждая сущность (строка) имеет уникальный идентификатор, называемый **первичным ключом.**
- Строка одной таблицы может быть связана с данными из другой при помощи **внешнего ключа.**

ИЗОЛЯЦИЯ

- **Изоляция** необходима для контроля над согласованностью и гарантирует независимость транзакций друг от друга.

НАДЕЖНОСТЬ

- **Надежность** подразумевает, что все внесенные в базу данных изменения на момент успешного завершения транзакции считаются постоянными.