



北京航空航天大學
BEIHANG UNIVERSITY

NoSQL简介及演示

林浩 博士



NoSQL简介

>>> NoSQL出现的原因

▪ 互联网、云计算与大数据背景下的系统设计需求

- 海量用户规模下的应用响应：低延迟的读写速度（频繁更新、大量读操作），提升用户体验
- 海量数据规模：PB级别的数据，需要稳定易用的扩展策略进行承载
- 半结构/非结构化数据存储：灵活可变的数据模型设计
- 庞大运营成本考量：硬件/软件/人力成本需大幅度地降低

▪ RDBMS的局限性

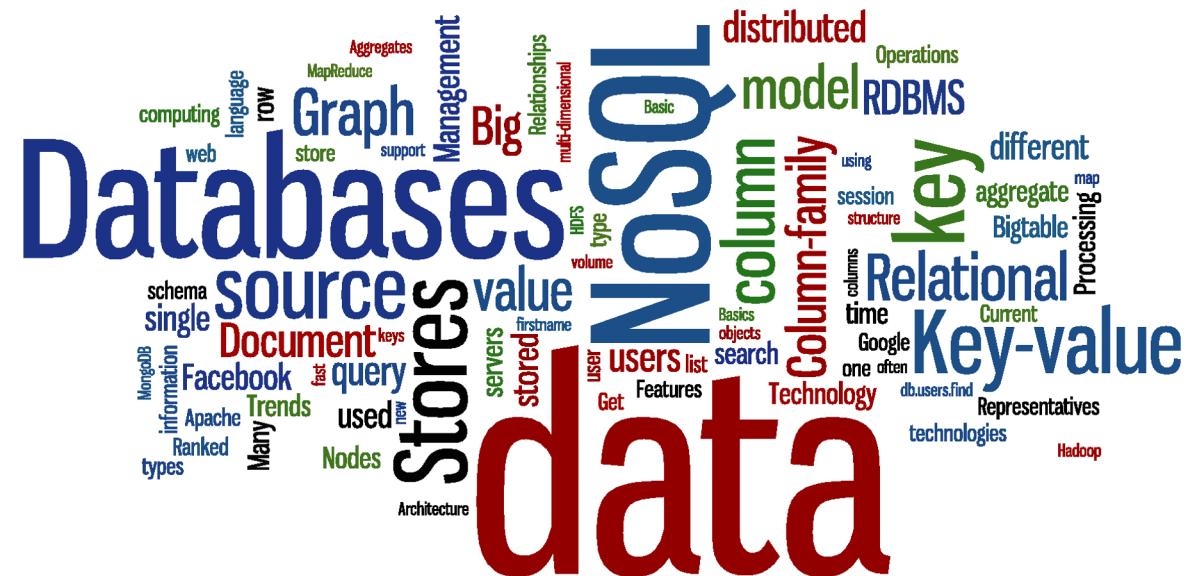
- 扩展困难：存在join等复杂查询功能，初始设计以单节点部署为主，同时单节点支撑容量有限
- 读写速度慢：数据规模达到一定程度后，极易发生死锁并发问题，需要复杂优化策略，维护成本高
- 部署成本高：企业级数据库SQL Server/Oracle价格昂贵（MySQL是开源的）

NoSQL (Not Only SQL)：相比于传统关系型数据库，在架构和数据模型方面适当做“减法”，在灵活性、扩展性和并发性等方面做“加法”，更关注海量数据存储、高并发读写与易用性

- 区别于传统关系型数据库设计的存储技术统称（缺乏统一定义）
 - 不使用传统的关系数据模型以及SQL查询语句
 - 模式自由：灵活的数据模型，记录（行）数据可相对自由地添加字段
 - 易用性：简单的查询方式
 - 分布式：设计在大型集群上运行（水平可扩展），CAP理论的最佳实践
 - 免费开源，初始开发时依靠开源社区支持
 - 海量数据规模下的优秀读写性能（尤其强调高并发下的读性能）、高可用性

▪ NoSQL可能存在的问题

- 技术成熟度
 - 客户支持几乎为零
 - 缺乏查询的统一标准



>>> NoSQL数据模型

- 四大类有别于传统关系型数据库的NoSQL数据模型

文档存储

- 比较适合存储系统日志、文本等非结构化数据
- 如CouchDB、MongoDB、Elasticsearch



mongoDB

键值存储

- 用哈希表维护Key 值到具体数据value的映射
- 适合通过主键进行查询或遍历，但对复杂的条件查询支持度不佳
- 如Redis、Dynamo、SimpleDB

Amazon DynamoDB

列族存储

- 基于Key-Value
- Value 具有更精巧的结构，即一个Value 包含多个列，列还可以分组
- 如BigTable、HBase、Cassandra

APACHE
HBASE

图存储

- 对大规模图数据进行有效的管理和分析
- 如Neo4j、Trinity



Neo4j
the graph database

- **数据的基本单元** : Document (文档) , 表示现实世界中的一个对象 , 一个文档对应 RDBMS 的一行 (Row) 数据
 - **文档** : 一组包含键值对 (key-value) 的层次化 JSON 结构体 (反范式设计)
 - 键 (key) 是字段 (field) 或属性 (property) 的名字 , 一般为字符串类型
 - 值 (value) 可以是字符串、数字、布尔类型、另一个对象、值数组或者其他特殊类型
 - **以 Elasticsearch (ES) 为例** , 文档中的键值对又可细分为文档元信息和文档数据

Elasticsearch中一条json格式的文档数据示例

```
{  
    "_index": "test", → 文档存储的索引名，对应RDBMS的数据库名  
    "_type": "weibo", → 文档表示的对象的类名，一组同类对象的类名，对应RDBMS的表名  
    "_id": "3978149677504320", → 文档的唯一标识符，对应RDBMS数据行的主键  
    "_source": {  
        "text": "巴塞罗那0:0塞尔维亚",  
        "user": { "user_fansnum": 1562, }  
        "user_id": 123  
    } }  
    } }  
    } }  
    } }
```

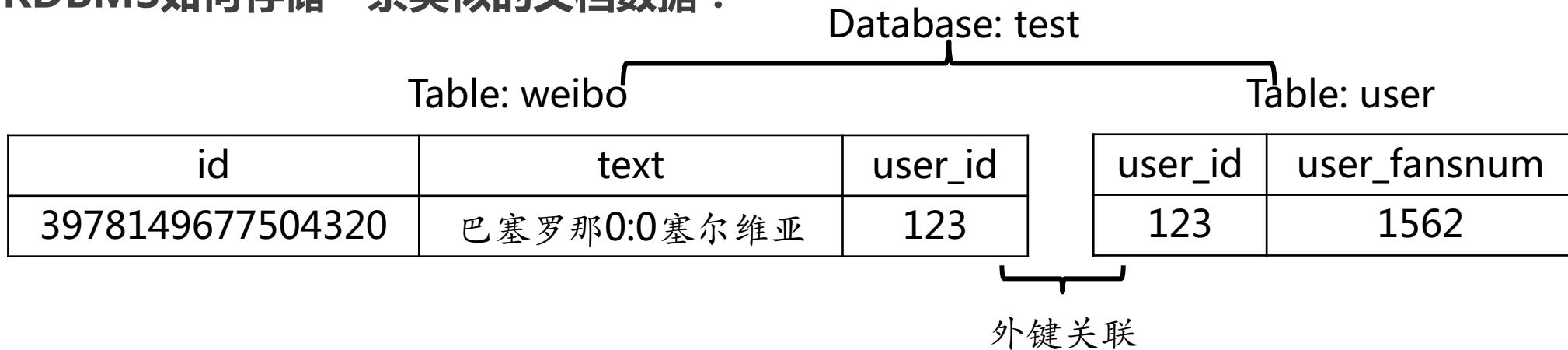
>>> NoSQL数据模型之文档存储



▪ MySQL与Elasticsearch的核心概念对应关系

MySQL	Elasticsearch
Database (数据库)	Index (索引)
Table (表)	Type (类型)
Row (行)	Document (文档)
Column (列)	Field (字段)
Schema (方案)	Mapping (映射)
Index (索引)	Everything Indexed by default (所有字段都被索引)
SQL (结构化查询语言)	Query DSL (查询专用语言)

▪ RDBMS如何存储一条类似的文档数据？



- **查询示例：查询发布过包含“巴塞罗那”关键词文本的用户粉丝数**
 - Elasticsearch：类似JSON方式的查询语句

```
query_body = {  
    "query":{  
        "match":{  
            'text': '巴塞罗那'  
        }  
    }  
}  
  
es = Elasticsearch()  
  
results = es.search(index="test", doc_type="weibo", body=query_body)["hits"]["hits"]  
for item in results:  
    print (item["_source"]["user"]["user_fansnum"])
```

- RDBMS：SQL join
- **NoSQL文档存储与RDBMS对比总结**
 - RDBMS使用表格结构（行、列），文档存储使用JSON结构，对事物建模更近于面向对象编程思想
 - 通过灵活的**反范式设计**，文档存储避免了在某些查询操作时RDBMS繁杂沉重的JOIN操作，对一个对象的表示更加紧凑（Self-contained），更易在单次操作中获得更多的字段
 - 文档存储可以允许不同的文档包含不同的字段，可以灵活动态地进行数据字段增减，字段值的类型也可以进行灵活变化

- 通过哈希表对数据进行存储，数据的基本单位是一个键值对，对应RDBMS中一行数据包含两列（ID主键列和数据列）
 - 以Redis为例，键一般为字符串类型，值的类型有很多可选项

Keys	Values
page:index.html	→ <html><head>[...]
login_count	→ 7464
users_logged_in_today	→ { 1, 4, 3, 5 }
latest_post_ids	→ [201, 204, 209,...]
user:123:session	→ time => 10927353 username => joe
users_and_scores	→ joe ~ 1.3483 bert ~ 93.4 fred ~ 283.22 chris ~ 23774.17

Redis键值存储的特点

- 查询方式：键值存储只能通过键来进行查询，无法通过值或值的属性来进行查询，值是不透明的
- 查询效率高：相较于RDBMS，按键索值的效率非常高，Redis每秒可以执行约110000个SET（按键写值），每秒执行约81000个GET（按键查值）
- 值的复杂数据结构支撑诸多应用
- 数据置于内存中，进一步加快读写速度

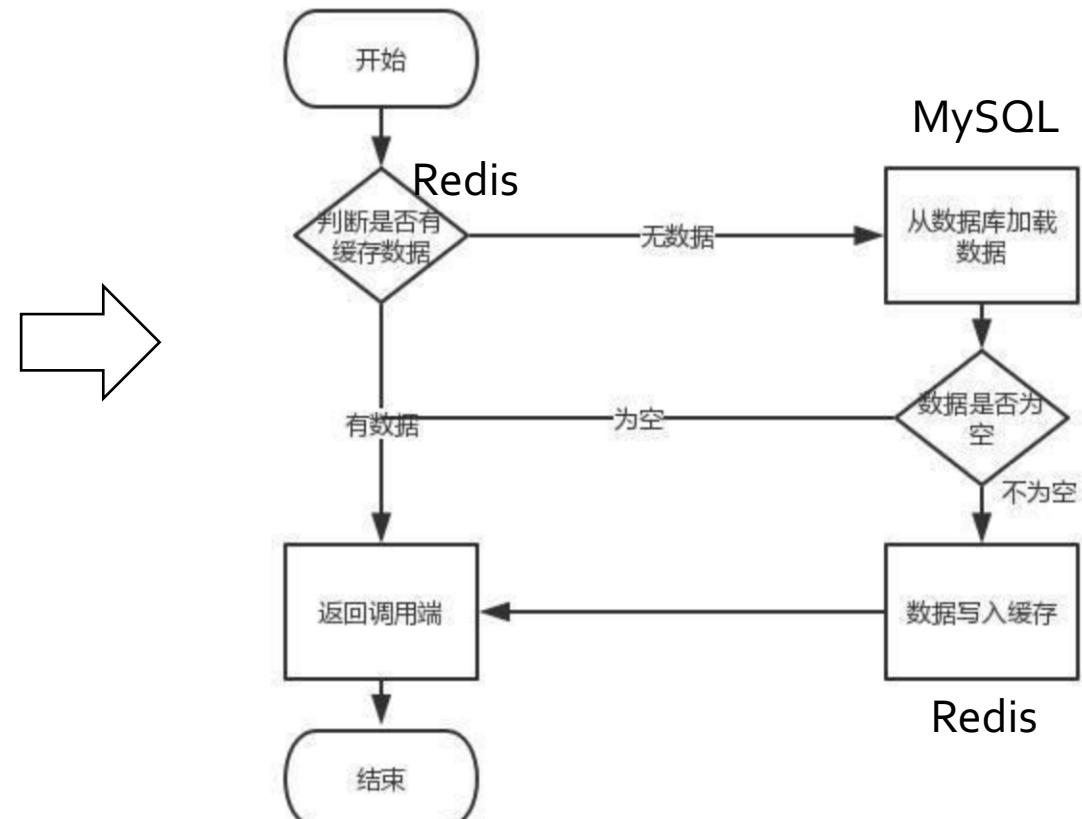
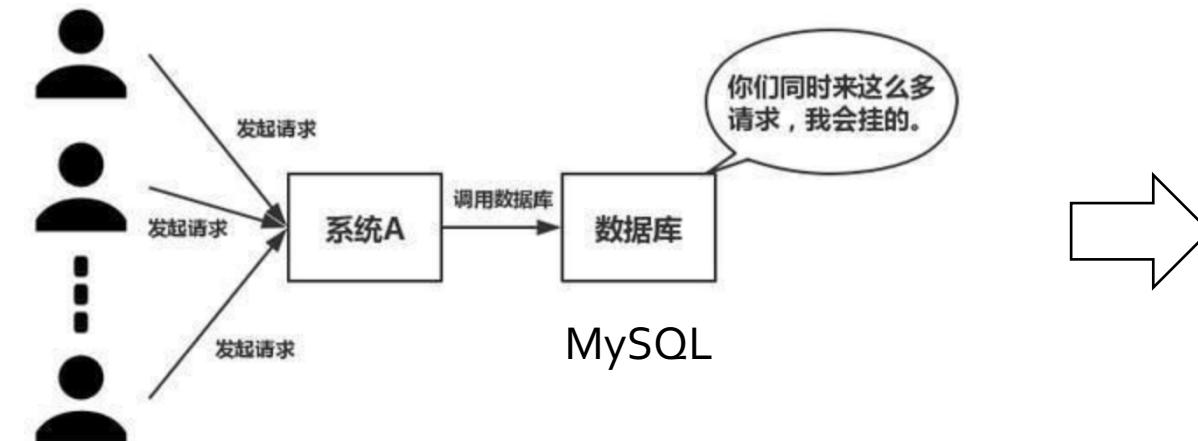
Redis等键值存储应用场景

- 缓存
- 实时计数
 - 存储网站Session会话数据、用户购物车数据
 - 存储用户偏好数据：user_id/user_name + 语言/时区/...
- 消息队列

>>> NoSQL数据模型之键值存储



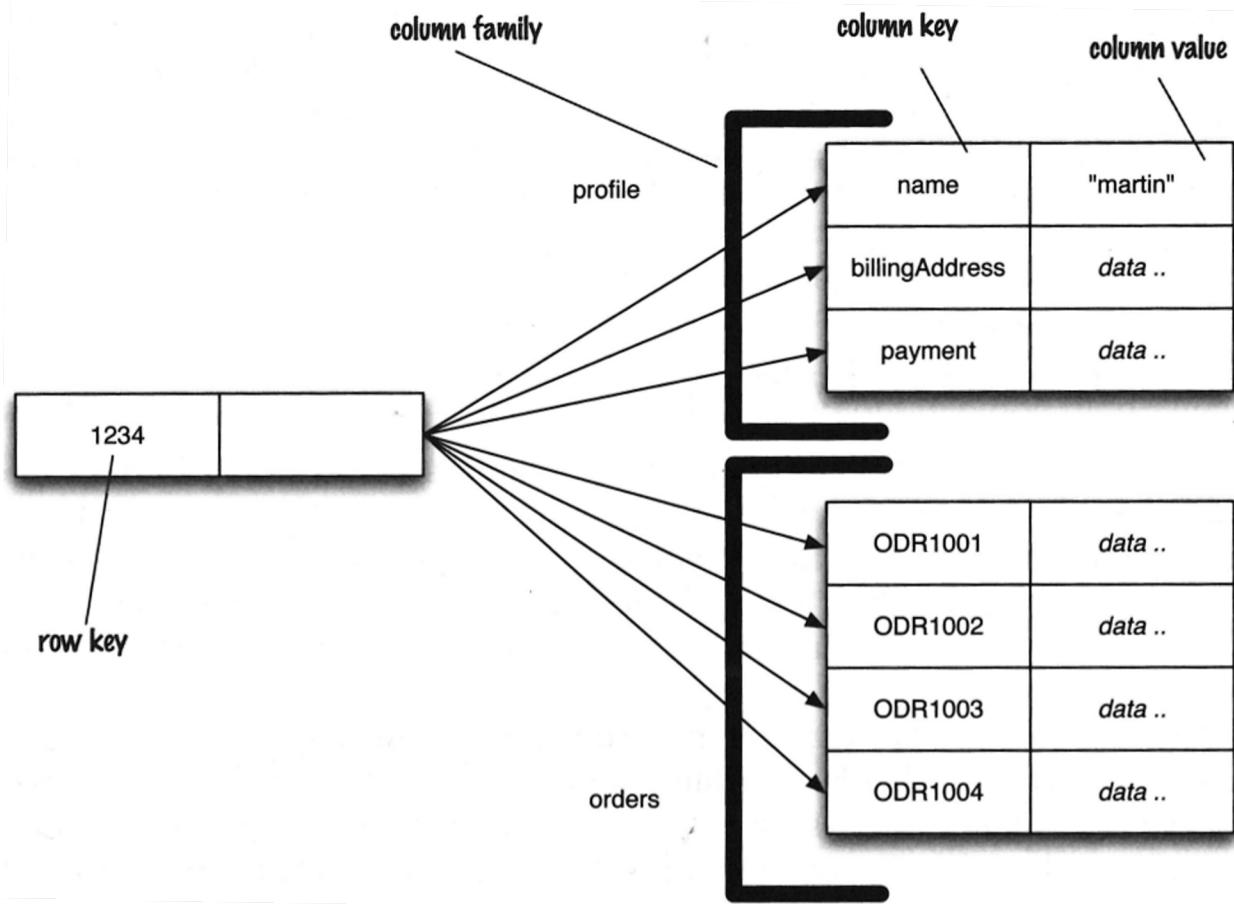
- Redis用于缓存数据库，与MySQL搭配使用，应对网站的高并发访问



>>> NoSQL数据模型之列族存储



- 列族存储：每行数据有一个行键（row key）和很多列（组成多个群组，称为列族Column Family），因此又名宽列（Wide Column）存储

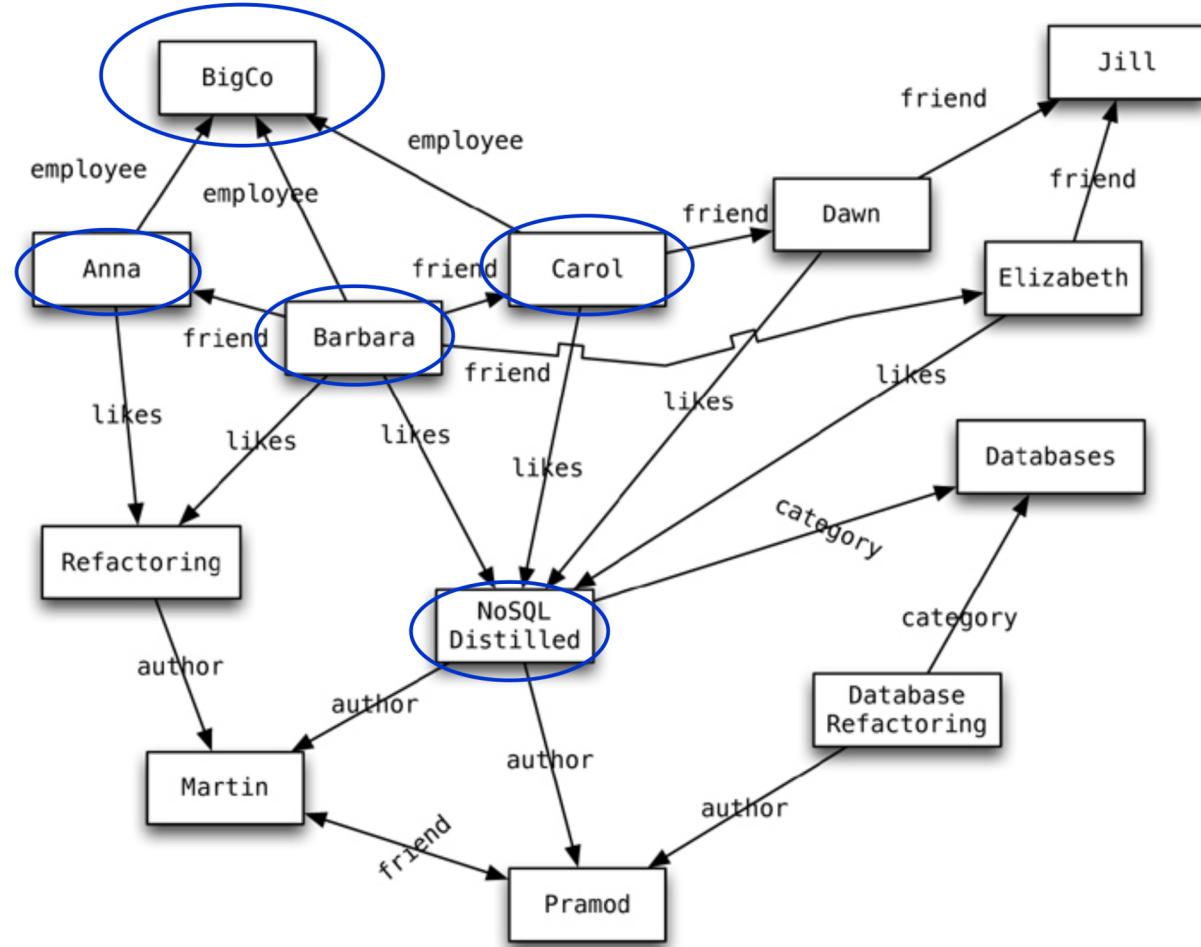


- **查询方式：**行键+列键定位数据
 - 相较于键值存储，在Value层次提供更细粒度的访问
- **列式存储**
 - 一列数据在存储介质中连续存储，返回众多行少量列的读取效率高
 - RDBMS是行存储，写入快，数据完整性高，适合少量随机读写以及返回所有属性的查询
- **列族作用**
 - 多个经常一起访问的数据列的各个值存放在一起，避免读取多列值再合并，提高查询效率
 - 例如，一些时候需要频繁地同时读取客户profile的多个字段（column）信息，另一些时候则只需要读取客户的所有订单信息

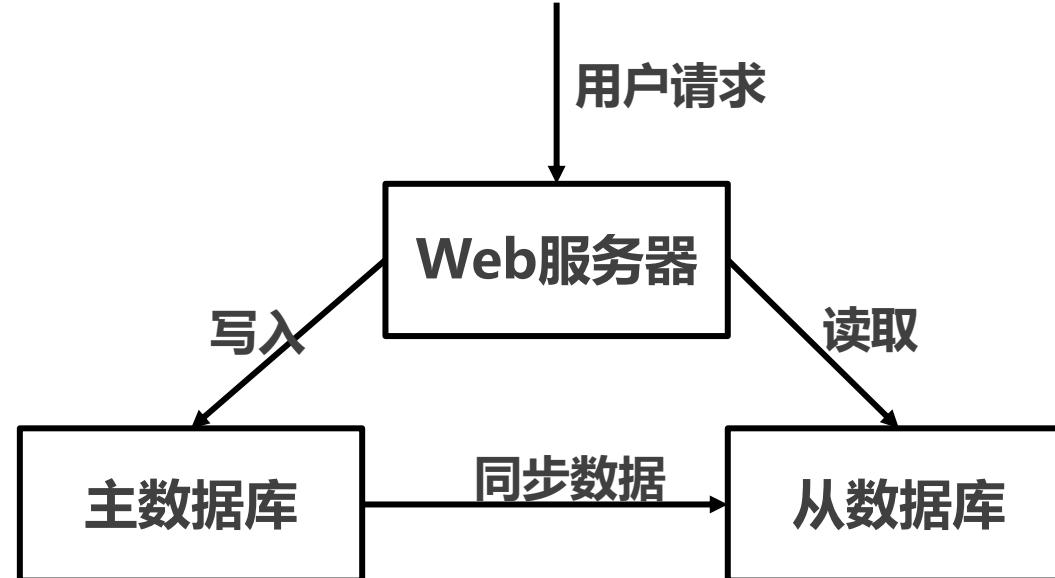
>>> NoSQL数据模型之图存储



- 图存储：以图结构表示真实世界中的数据以及数据之间的关系
 - 节点：对象实例
 - 节点属性：对象的属性
 - 边：对象之间的关系，具备方向性和类型
- 图存储优势
 - 高效的图遍历性能：查询所有喜欢NoSQL Distilled的BigCo公司的雇员
 - 非结构化的存储形式：灵活与伸缩性
 - 数据模型简单直观：物理世界存在形式与数据库存储方式非常接近
- RDBMS如何存储图数据？
 - 一张表只能存储一种关系，增加关系只能修改模式定义
 - 定义模式时需要提前考虑下游的图遍历的应用需求



- CAP = **Consistency** (一致性), **Availability** (可用性), **Partition tolerance** (分区容忍性)



- ✓ C : 写操作后的读操作可以读取到最新的数据状态，当数据分布在多个节点上，从任意结点读取到的数据都是最新的状态。由于数据写入/同步存在网络延迟，可通过在从数据库上加读锁，实现强一致性，即任意时刻从数据库返回的数据要么提示错误（未同步完成），要么返回与主节点一致的最新数据。
- ✓ A : 任何时间访问系统，都不会出现响应超时或错误，即读取从数据库时，不能加读锁；即使没有同步完成，也得返回旧数据（或默认值），不能返回错误。
- ✓ P : 同步数据失败（网络失败或超时过久）不能影响读写，从节点挂掉不影响读操作，可通过添加更多的从节点实现，并且同步操作是异步执行的。这是分布式系统的基本要求。

CAP : Brewer提出、Gilbert和Lynch证明，在大型**分布式**系统中，**一致性** (Consistency)、**可用性** (Availability)、**分区容忍性** (Partition tolerance) 三个目标中，同时只能实现两种特性



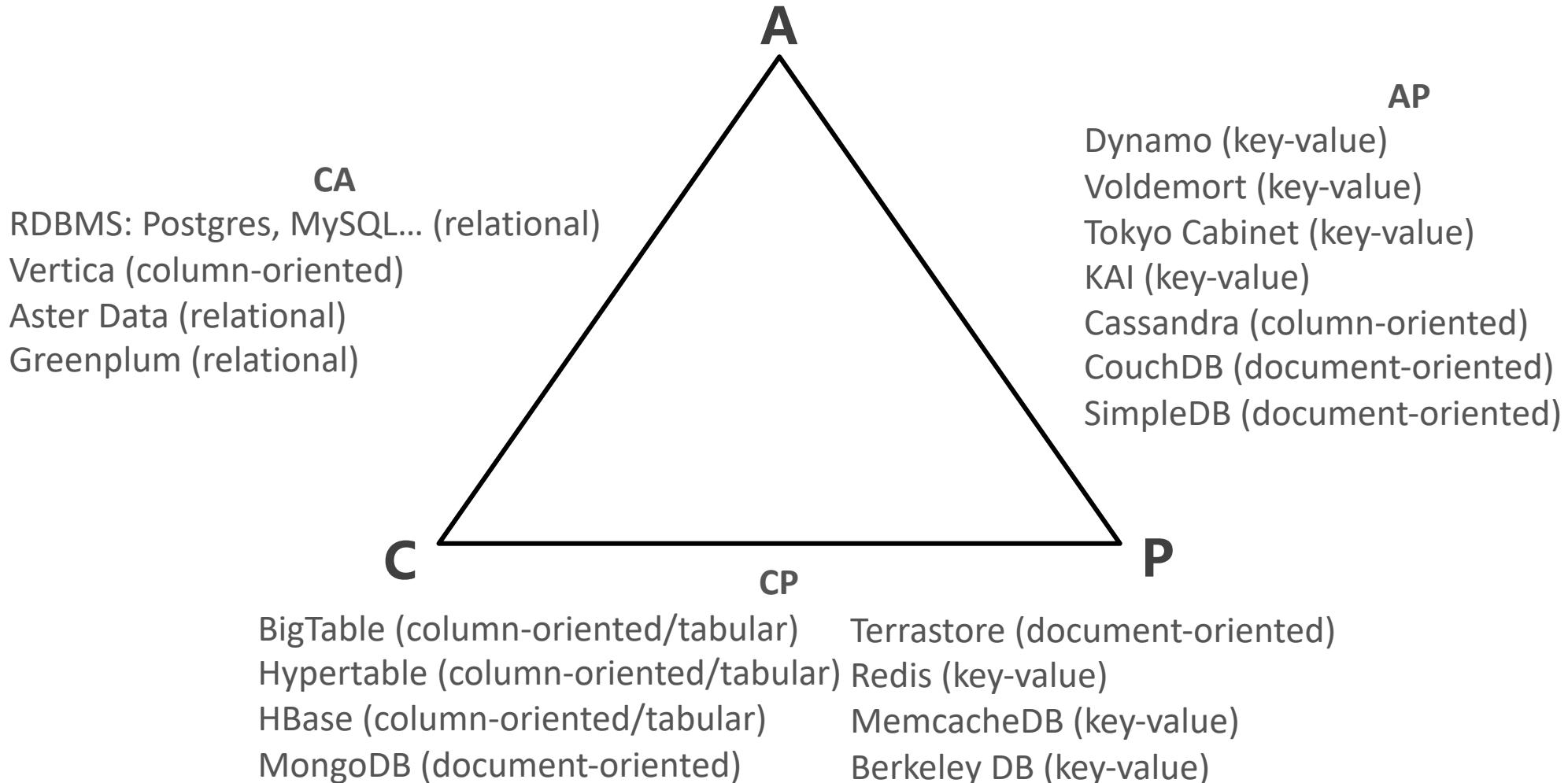
RDBMS : 一般为单机CA系统，追求系统高度一致性和可用性

NoSQL

- ✓ 由于需要通过横向扩展的方式提供扩展存储能力，一般为分布式系统
- ✓ 分布式条件下，需优先保证分区容忍性P
- ✓ 分布式条件下，C和A是互斥的，可选择放松一致性或可用性要求，形成AP或CP系统

- AP : 放松强一致性要求，接受最终一致性，订单退款，今日退款成功，明日账户到账，只要用户可以接受在一定时间内到账即可。
- CP : 放弃可用性，一次转账请求要等待双方银行系统都完成整个事务才算完成。

- 根据CAP理论对NoSQL进行分类



参考Visual Guide to NoSQL Systems进行绘制

>>> NoSQL发展态势：混合多模态存储



▪ 多种NoSQL技术混合：Facebook

- Apache Hbase : 邮件、即时消息、短信息
- Memcached : Web服务器与MySQL服务器之间的缓存
- Apache Giraph : 存储用户关系的图数据库
- RocksDB : 高性能键值数据库

APACHE
HBASE



▪ NoSQL与RDBMS混合：亚马逊电子商务应用

- 最新的产品和订单信息 : RDBMS
- 购物篮数据 : 键值存储
- 大量历史订单信息 : 类似MongoDB的文档存储

NoSQL演示

- 演示目标

- 以当下最热门的NoSQL文档存储（分布式实时搜索引擎）Elasticsearch（简称ES）和最热门的开源RDBMS MySQL为数据库工具，在示例数据集上，分别演示利用Python代码对NoSQL和RDBMS进行数据的写入和读取（查询）的过程，并简单对比其性能

- 演示流程

- Step 1：安装并启动ES服务（在一台PC机部署两个ES instance，构成功能性集群）和MySQL服务
- Step 2：创建ES索引结构和MySQL表结构
- Step 3：分别向ES、MySQL写入数据，并对比写入速度
- Step 4：分别从ES、MySQL进行全文检索，并对比检索和数据读取速度

注：演示操作会有更详细说明在“[NoSQL 演示具体步骤说明.pdf](#)”，相关代码和数据在code文件夹中，都会发放给同学们

▪ 总体要求

- 按照本讲演示的思路，每个同学在本地计算机上独立完成示例数据集（news.txt）上ES和MySQL数据写入、查询过程，对比两种数据库的功能和性能差异，并撰写实验报告，实验报告内容应包括本机上相关软件安装过程说明与截图、相关Python代码及说明、Python代码在本机运行结果（程序输出和效率）截图以及其他必要的说明
- 提交方式：将实验报告提交给助教，提交截止时限为本讲结束后一个月内

▪ 具体操作内容除复现课堂演示的相关内容（基本要求）外，还可包括（下面为额外内容）：

- 基于示例数据集news.txt，模拟生成不同数量级的人工数据集（比如复制示例数据集若干份，同时修改每条数据的唯一标识符），作为输入数据，对比不同数据规模下MySQL和ES的写入和读取速度，可利用matplotlib等工具绘制对比散点图。（**拓展要求，不做硬性要求**）
- 以示例数据或模拟生成的某个规模人工数据集（**建议使用较大规模数据，可能差别更显著**），作为输入数据，对比不同集群节点数下ES的写入和读取速度。课程演示的是单机双节点ES，同学们可根据自己电脑内存配置情况，可配置单机3节点、4节点或更多节点的ES，组成更大规模集群。（**拓展要求，不做硬性要求**）
- 可参考ES官网资料，尝试修改配置或修改查询语句，提升ES写入或查询速度。（**拓展要求，不做硬性要求**）
- 查阅相关资料，安装和配置MongoDB，将其加入与ES、MySQL的读写性能对比测试中（**拓展要求，不做硬性要求**）

注：鉴于除课堂演示内容外的其他要求对coding有一定的要求，不做硬性要求，但鼓励大家进行尝试。作业仍以课堂演示的相关内容为主。