非关系数据库第一次作业-10205101530-赵晗瑜

1.（20’）简述大数据的基本特征有哪些？请举例3个大数据应用的场景，并介绍和分析应用场景中的数据。

大数据的基本特征：

- 大量（**Volume**）：数据量大TB->PB，即大数据的起始计量单位至少是P（1000个T）、E（100万个T）或Z（10亿个T）。
- 高速（**Velocity**）：数据被创建，移动和处理的速度高；处理速度快，时效性要求高。这是大数据区分于传统数据挖掘最显著的特征。
- 多样（**Variety**）：即数据类型繁多。这种类型的多样性也让数据被分为结构化数据和非结构化数据。相对于以往便于存储的以文本为主的结构化数据，非结构化数据越来越多，包括网络日志、音频、视频、图片、地理位置信息等，这些多类型的数据对数据的处理能力提出了更高要求。
- 价值（**Value**）：具有价值，但价值密度低。如随着物联网的广泛应用，信息感知无处不在，信息海量，但价值密度较低，如何通过强大的机器算法更迅速地完成数据的价值“提纯”，是大数据时代亟待解决的难题。

三个大数据应用的场景：

1. 大数据应用案例之社交网络，如**LinkedIn**

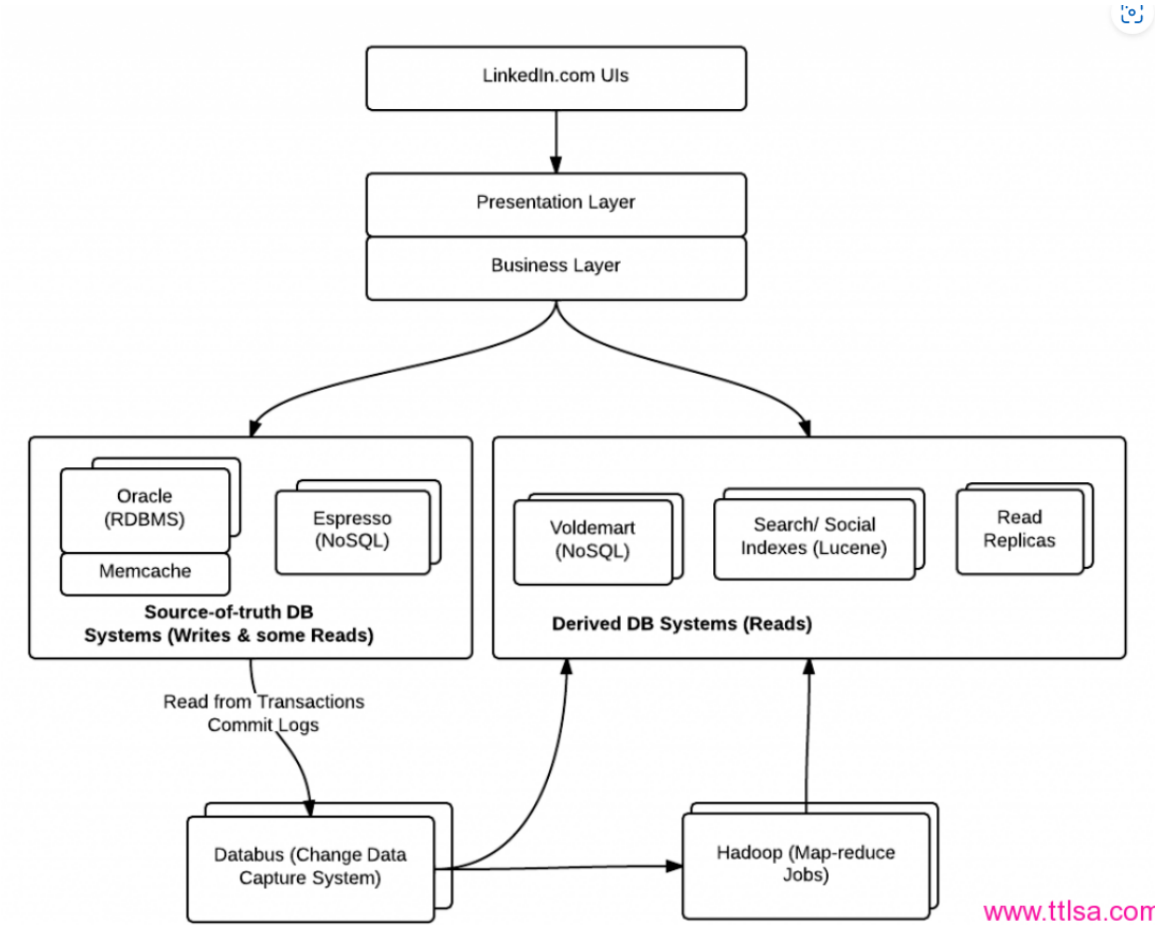
   网站的目的是让注册用户维护他们在商业交往中认识并信任的联系人，俗称“人脉”。因此，网站需要存储大量用户的数据及其用户之间的关系。

   LinkedIn工程师团队提出了三段式（**three-phase**）数据架构，由在线，离线以及近线数据系统组成。总体上讲，**LinkedIn**数据被存储在如下几种不同形式的数据系统中。

   - RDBMS
     - Oracle
     - MySQL（作为Espresso的底层数据存储）
   - RDBMS
     - Espresso（LinkedIn自己开发的文档型NoSQL数据存储系统）
     - Voldemart （分布式Key-value存储系统）
     - HDFS （存放Hadoop map-reduce任务的数据）

- Caching
    - Memcached
- 基于Lucene的索引
    - 存放查询、关系图等功能数据的Lucene 索引
    - Espresso使用的索引

如下图，LinkedIn数据库系统包括了DataBus、NoSQL、RDBMS以及Indexes



由**RDBMS**，**NoSQL**等构成。其中**Espresso**是一个支持水平扩展、索引、时间线一致性、基于文档且高可用的**NoSQL**数据仓库，旨在代替支撑公司网页操作所使用的传统**Oracle**数据库。**Espresso** 具有一个分层的数据模型，其结构为数据库（**Database**）-> 表（**Table**）> 集合（**Collection**）-> 文档（**Document**）。**Espresso** 使用 **JSON** 数据格式定义数据库 **Schema** 和表 **Schema**，使用数据序列化的系统 **Apache Avro** 定义文档 **Schema**。

LinkedIn所使用的三段式数据架构分为在线数据库系统，离线数据库系统和近线数据库系统（时间线一致性）：

- 在线数据库系统

    在线数据库系统处理与用户的实时互动。主数据存储用来支撑用户的写操作和少量的读操作。如oracle，Oracle会执行所有的写操作。最近，LinkedIn正在开发另一个叫做“Espresso”的数据系统来满足日益复杂的数据需求，而这些数据看似不应从像Oracle这类的RDBMS中获取。

Espresso是一个支持水平扩展、索引、时间线一致性、基于文档且高可用的NoSQL数据仓库，旨在代替支撑公司网页操作所使用的传统***Oracle***数据库。设计它的初衷是为了提高LinkedIn的InMail消息服务的可用性。目前有如下一些应用在使用Espresso作为可信源系统

  - 成员间消息,
  - 社交动作，如：更新
  - 文章分享
  - 用户个人资料
  - 公司资料
  - 新闻文章
- 离线数据库系统

  离线系统主要包括Hadoop和一个Teradata数据仓库，用来执行批处理和分析类的工作。之所以被称为离线是因为它对数据执行的的批处理操作。Apache Azkaban被用来管理Hadoop和ETL任务，这些任务从主可信源系统获取数据后交由map-reduce处理，处理结果被保存在**HDFS**，然后通知'消费者'（例如：**Voldemart**）通过合适的方式来获取这些数据并切换索引来保证能获取到最新的数据。
- 近线数据库系统（时间线一致性）

  近线系统的目标是为了实现时间线一致性（或最终一致性），它处理类似'你可能认识的人（只读数据集）'、搜索以及社交图这些功能，这些功能的数据会持续更新，但它们对延迟性的要求并不像在线系统那样高。

用数据用例来展示它们是如何工作的

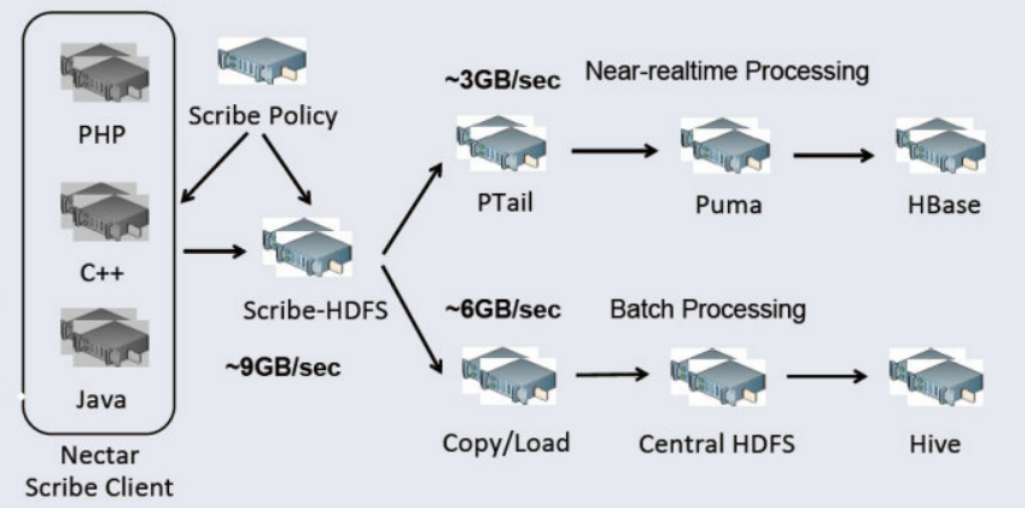
假如你更新了你个人资料中的最新技能和职位。你还接受了一个连接请求。那么在系统内部到底发生了什么：

- 将更新写入Oracle Master数据库
- 然后Databus做了如下一系列奇妙的工作来实现时间线一致性：
  i. 将资料变更，如最新技能和职位信息，更新到标准化服务。
  ii. 将上面提到的变更更新到搜索索引服务。
  iii. 将关系变更更新到图索引服务。

2. 大数据应用案例之**facebook**

- 数据量大：资料显示2011年它拥有的压缩数据已经有25PB，未压缩数据150PB,每天产生的未压缩的新数据有400TB
- 应用领域：大数据技术被广泛应用在广告、新闻源、消息/聊天、搜索、站点安全、特定分析、报告等各个领域
- 数据架构：

  Facebook当前的大数据技术架构是在早期架构基础上对数据传输通道和数据处理系统进行了优化，如图所示，主要分为分布式日志系统**Scribe**、分布式存储系统**HDFS**和**HBase**、分布式计算和分析系统（**MapReduce**、**Puma**和**Hive**）等。



如上图：

i. **Scribe**：Scribe日志系统用于聚合来自大量HTTP服务器的日志数据

ii. **Thrift**：Thrift是Facebook提供的软件框架，用于跨语言的服务开发，能够在C 、Java、PHP 、Python和Ruby等语言之间实现无缝的支持

iii. **Thrift RPC**：采用Thrift RPC来调用Scribe日志收集服务进行日志数据汇总。

iv. **Scribe Policy**：Scribe Policy是日志流量和模型管理节点，将元数据传送给Scribe客户端和Scribe HDFS，采集的日志数据存储在Scribe HDFS

v. **Data Freeway**：Facebook对早期系统优化后的数据通道，Data Freeway主要包括4个组件，Scribe、Calligraphus、Continuous Copier和PTail

i. Scribe用于客户端，负责通过Thrift RPC发送数据

ii. Calligraphus在中间层梳理数据并写到HDFS，它提供了日志种类的管理，利用Zookeeper进行辅助

iii. Continuous Copier将文件从一个HDFS拷贝到另一个HDFS

iv. PTail并行地tail多个HDFS上的目录，并写文件数据到标准输出

vi. 在当前架构中，一部分数据处理仍然以批处理的方式通过MapReduce进行小时级的处理，存储在中央的HDFS，每天通过Hive进行分析处理。另一部分接近实时的数据流则通过Puma来进行分钟级的处理。Facebook对专门分析提供Peregrine（Hipal）工具、对周期性分析提供Nocron工具进行分析。

3. 大数据应用案例之搜索引擎：**google，baidu**

以Google为例，需要存储大量**Web**文档。Google 使用的大数据平台主要包括 3 个相互独立又紧密结合在一起的系统：

- Google 文件系统（Google File System，GFS）。

> Google File System(简称GFS)是适用于大规模且可扩展的分布式文件系统，可以部署在廉价的商务服务器上，在保证系统可靠性和可用 性的同时，大大降低了系统的成本。以这种方式写的程序能自动的在大规模的普通机器上实现并行化

- 针对 Google 应用程序的特点提出的 MapReduce 编程模式

> MapReduce是一个编程模型,和处理,产生大数据集的相关实现。用户指定一个map函数处理一个key/value对,从而产生中间的key/value对集.然后再指定一个reduce函数合并所有的具有相同中间key的中间value

- 大规模分布式数据库 BigTable。

> BigTable 是 Google 设计的分布式数据存储系统，是用来处理海量数据的一种非关系型数据库。BigTable 是一个稀疏的、分布式的、持久化存储的多维度排序的映射表。

2.（20’）简述非关系型数据库出现的背景，区别非关系型数据库、NoSQL数据库以及NewSQL数据库。

非关系数据库出现的背景：

首先，传统关系数据库自身的局限性无法应对大数据的应用需求，如扩展困难，读写慢，成本高，容量有限等；为了应对数据高并发的读写需求和海量数据的存储需求，非关系型数据库应用而生。

> 1、High performance--对数据库高并发读写的需求
>
> 2、Huge Storage--对海量数据的高效率存储和访问的需求
>
> 3、High Scalability && High Availability--对数据库的高可扩展性和高可用性的需求

在上面提到的“三高”需求面前，关系数据库遇到了难以克服的障碍，而对于web2.0网站来说，关系数据库的很多主要特性却往往无用武之地，例如：

1.数据库事务一致性需求

很多web实时系统并不要求严格的数据库事务，对读一致性的要求很低，有些场合对写一致性要求也不高。

2.数据库的写实时性和读实时性需求

对关系数据库来说，插入一条数据之后立刻查询，是肯定可以读出来这条数据的，但是对于很多web应用来说，并不要求这么高的实时性。

3.对复杂的**SQL**查询，特别是多表关联查询的需求

任何大数据量的web系统，都非常忌讳多个大表的关联查询，以及复杂的数据分析类型的复杂SQL报表查询，特别是SNS类型的网站，从需求以及产品设计角度，就避免了这种情况的产生。往往更多的只是单表的主键查询，以及单表的简单条件分页查询，SQL的功能被极大的弱化了。

因此，关系数据库在这些越来越多的应用场景下显得不那么合适了，为了解决这类问题的非关系数据库应运而生。

非关系数据库，**NoSQL**与**NewSQL**的区别：

英文名	中文名	定义	存储方式	**ACID**规则支持情况	**CAP**原理支持情况

英文名	中文名	定义	存储方式	**ACID**规则支持情况	**CAP**原理支持情况
Relational database	关系型数据库	采用了关系模型来组织数据的数据库	表格	支持ACID原则	满足CP，但A不完美
NoSQL	非关系数据库	泛指非关系型数据库，不保证关系数据的ACID特性	键值存储，列存储，文档存储等	不一定完全支持	满足AP，但C不完美
NewSQL	NewSQL	NewSQL是对各种新的可扩展/高性能数据库的简称	多种存储方式	支持	满足CAP

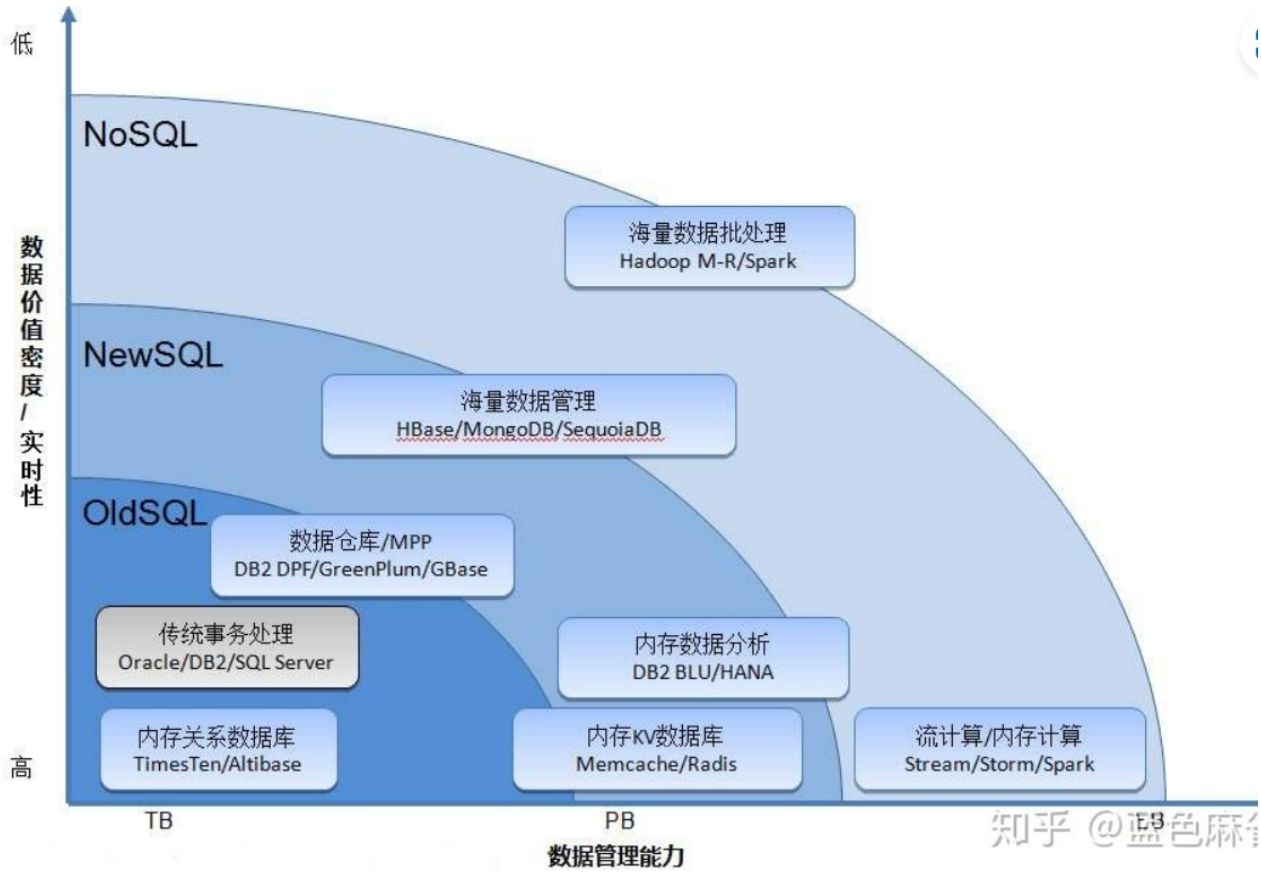
	SQL	**NOSQL**	**NEWSQL**
关系模型	√	×	√
SQL语句	√	×	√
ACID	√	×	√
水平扩展	×	√	√
大数据	×	√	√
无结构化	×	√	×

在数据容量上：

传统关系型数据库的数据容量有限，处理性能有限，可扩展性也有限

Newsql介于传统数据库和nosql之间，即支持关系型数据查询，又能存储更大容量的数据

Nosql最大的特点就是支持非结构化数据，更大容量的存储



关系型数据库

关系型数据库，是指采用了关系模型来组织数据的数据库

1. 关系模型指的就是二维表格模型，而一个关系型数据库就是由二维表及其之间的联系所组成的一个数据组织。
2. 通过**SQL**结构化查询语句存储数据。
3. 强调**ACID**(原子性，一致性，隔离性，持久性)规则, 保持数据一致性。这些准则保证了数据库尤其是每个事务的稳定性，安全性和可预测性。
4. SQL的主要问题是它难以扩展，因为它的性能随着数据库的变大而快速下降。 分布式也是有问题的。

非关系数据库（**NoSQL**）

1. 非关系型数据库又被称为 NoSQL（Not Only SQL），意为不仅仅是 SQL。通常指数据以对象的形式存储在数据库中，而对象之间的关系通过每个对象自身的属性来决定，常用于存储非结构化的数据。
2. Not Only SQL符合非关系型、分布式、开源和具有水平可扩展能力的下一代数据库。NoSQL以放宽**ACID**原则为代价，采取最终一致性原则，而不是像关系型数据库那样地严格遵守着ACID的原则。
3. 非关系型数据库的优势：

a. 在支持的数据类型上：非关系型数据库存储数据的格式可以是 **key-value** 形式、文档形式、图片形式等。使用灵活，应用场景广泛，而关系型数据库则只支持基础类型。
b. 在处理速度上：速度快，效率高。 NoSQL 可以使用硬盘或者随机存储器作为载体，而关系型数据库只能使用硬盘。
c. 海量数据的维护和处理非常轻松，成本低。
d. 非关系型数据库具有扩展简单、高并发、高稳定性、成本低廉的优势。
e. 可以实现数据的分布式处理。

非关系型数据库存在的不足：

a. 非关系型数据库暂时不提供 **SQL** 支持，学习和使用成本较高。
b. 非关系数据库没有事务处理，无法保证数据的完整性和安全性。适合处理海量数据，但是不一定安全。
c. 功能没有关系型数据库完善。
d. 复杂表关联查询不容易实现。

NewSQL

1. NewSQL是一种相对较新的形式，旨在使用现有的编程语言和以前不可用的技术来结合**SQL**和**NoSQL**中最好的部分。 NewSQL目标是将SQL的**ACID**保证与**NoSQL**的可扩展性和高性能相结合。

3.（10’）给出自己对Multi-Model Database的理解

简单来说，原生多模型数据库就是将多种数据存储组合在一起。结构化数据、半结构化数据和非结构化数据往往以不同的格式和模型存储，而多模态数据库系统尝试用单一的数据库平台来管理这些数据。既然是不同的数据类型，那么为什么不采用不同的数据库呢？比如结构化数据采用MySQL数据库，半结构化数据采用NoSQL数据库呢？

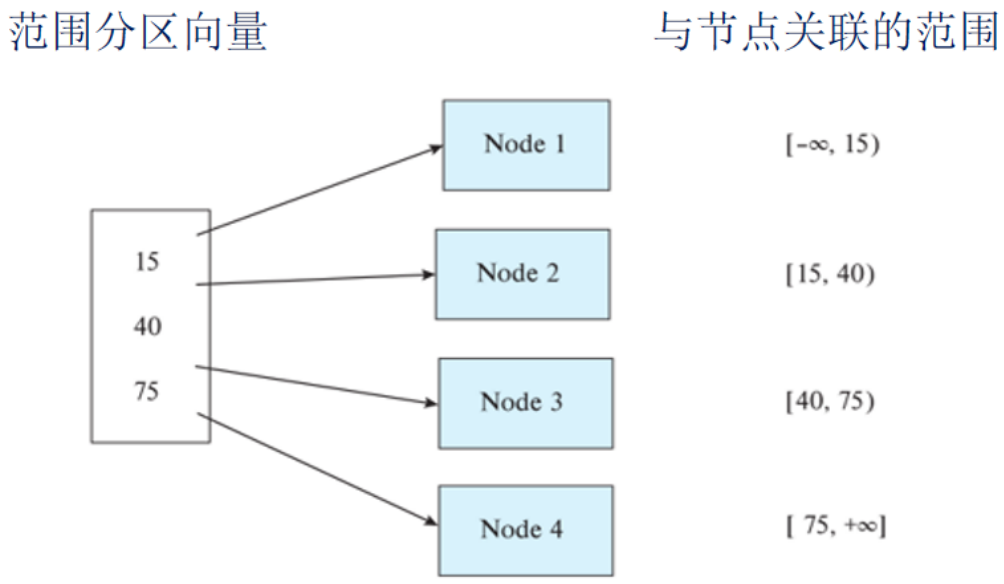
原因是：在同一个项目中采用多种数据库，这也就引入了运维的复杂和繁琐（部署更复杂，升级更频繁），数据一致性问题和数据冗余问题。而多模型数据库则是在一个数据库中，可以同时支持多种存储引擎或存储类型，为应用提供各种数据服务。多模型数据管理能力，使得数据库能够进行跨部门，跨业务的数据统一存储与管理，实现多业务数据融合，支撑多样化的数据服务。

4.（20’）简述I/O并行化的几种技术

I/O并行是指通过在多个节点（计算机）上对多个磁盘上的数据集进行分区，减少从磁盘检索数据所需的时间。

I/O并行技术**(**设节点数为**n):**

1. 轮询方法（Round-robin）
    a. 第i条记录存储到的节点为 i mod n.
2. Hash分区
    a. 选择一个或多个属性作为分区属性
    b. 选择取值范围为0...n-1的哈希函数h
    c. 设i 为哈希函数h应用于记录属性的计算结果，然后将记录存储在节点i
3. 范围分区
    a. 选择分区的属性
    b. 选定分区向量$vector[V_0, V_1, \ldots, V_{n-2}]$
    c. 设v是一个记录分区属性的值，那么$V_i <= V_{i+1}$ 的记录分配到节点 i + 1；$V < V_0$ 的记录分配到节点0；$V >= V_{n-2}$ 的记录分配到节点n-1
    d.



4. 分片技术
    a. 水平地将大的数据集划分成较小的、易于管理的数据集的过程。
    b. 每个小数据集可以独立地为所负责的数据提供读写服务
    c. 某个查询的数据可能来自两个小数据集
    d. 数据分片要考虑查询模式以便小数据集本身不会成为性能瓶颈
5. 复制技术
    a. 多个节点上存储数据集的多个拷贝，称作副本
    b. 相同的数据在不同的节点上存在多个副本，提供了可伸缩性、可用性和容错性
    c. 复制实现方法：主从复制&对等复制

5.（30’）简述事务的定义及特征，介绍基于时间戳和基于多版本的并发控制机制

事物的定义和特征：

定义：

事务是描述完成数据处理的完整逻辑工作单元的操作集合。

在关系数据库管理系统中，为了保证数据一致性，数据库系统中事务必须具有ACID特性：

特性：

一致性（consistency）：以隔离方式执行事务，（即，没有其他事务的并发执行）以保持数据库的一致性。确保单个事务的一致性是编写该事务的应用编程人员的责任

原子性(Atomicity)：事务的所有操作在数据库中要么全部正确反映出来，要么完全不反映。如果一个事务从未开始或保证完成，那么除了在该事务的执行期间，不一致状态应该是不可见的，而这是需要原子性的原因，如果有原子性，那么事务的所有操作要么在数据库中完全反映出来，要么完全不反映。保证原子性是数据库的责任。

隔离性(isolation)：尽管多个事务可能并发执行，但系统保证，对于任何一对事务Ti和Tj,在Ti看来，Tj或者在Ti开始之前已经完成执行，或者在Ti完成之后开始执行。因此每个事务都感觉不到系统中有其他事务在并发地执行。事务的隔离性确保事务并发执行所得到的系统状态与这些事务以某种次序一次执行一个后所得到的状态是等价的。确保隔离性是数据库系统中称作并发控制系统的部件的责任。

持久性(durability)：一个事务完成成功后，它对数据库的改变必须是永久的，即使出现系统故障

基于时间戳的并发控制机制

基于锁的并发控制协议在事务集合运行过程中，根据事务对数据对象的加锁顺序动态决定事务pair中事务调度顺序。而基于时间戳的并发控制协议则根据事务的时间戳事先决定事务的调度顺序。

1.概念

每个事务有一个时间戳TS(Ti)，时间戳的实现方式：

- 系统时钟

- 逻辑计数器

在基于时间戳的并发控制协议中，时间戳决定了事务的调度顺序，决定了事务对同一个冲突数据对象的访问顺序，每个数据对象也有两个时间戳：

- W-timestamp(Q)：最近完成对Q写的事务的时间戳
- R-timestamp(Q)：最近完成对Q读的事务的时间戳

2.T/O协议内容

1. 事务的READ(Q)读操作：
    - 如果TS(Ti) < W-timestamp(Q)，说明数据已经被更新的事务覆写，当前事务回滚。
    - 如果TS(Ti)>= W-timestamp(Q)，说明当前读的数据对象是合法的，读操作执行，然后置R-timestamp(Q)=max(R-timestamp(Q)，TS(Ti))
2. 事务的WRITE(Q)写操作：
    - 如果TS(Ti) < R-timestamp(Q)或者TS(Ti) < W-timestamp(Q)，说明有更性的事务对Q进行了读或写操作，因此当前事务被回滚
    - 反之，则当前事务写执行成功，并将W-timestamp(Q)设置为TS(Ti)。

事务回滚后，系统将给事务重新分配时间戳。

T/O协议可以避免死锁的产生（破坏了死锁的请求和等待条件），但是可能导致事务饿死（长执行时间事务被不断回滚）。需要采取额外的措施避免事务饿死。

原始的T/O并发控制协议是冲突可串性化的调度，但是不能保证可恢复性和无级联性。扩展T/O协议支持：

- 可恢复性：追踪事务所有访问对象，当所有对事务读对象进行修改的事务提交后再进行提交（困难）
- 无级联性：事务所有写操作归集到事务末尾执行 或者 事务对未提交数据的读延迟至对该数据写事务完成提交。

3.性能改进的**T/O**协议**--Thomas' Write Rule**

主要思想：旧时间戳事务的写操作在特定场景下被忽略掉，可以视为此事务的写被其他事务覆盖写了。

协议在原生T/O协议的基础上，修改事务的写操作WRITE(Q)逻辑：

1. 如果TS(Ti) < R-timestamp(Q)，说明旧值被新事务读取，因此当前事务写操作不可执行，事务被回滚。

2. TS(Ti) < W-timestamp(Q)，说明有新事务对值进行写，因此当前事务的写操作可以被忽略。
3. 此时，TS(Ti) >= R-timestamp(Q)且TS(Ti) >= W-timestamp(Q)，因此事务可以执行WRITE(Q)操作，并且置W-timestamp(Q)=TS(Ti)

基于多版本的并发控制机制

1.定义

MVCC（Multi-Version Concurrency Control，多版本并发控制）一种并发控制机制，在数据库中用来控制并发执行的事务，控制事务隔离进行。

2.核心思想：

MVCC是通过保存数据在某个时间点的快照来进行控制的。也就是说，不管事务执行多长时间，事务内部看到的数据是不受其它事务影响的，根据事务开始的时间不同，每个事务对同一张表，同一时刻看到的数据可能是不一样的。使用MVCC就是允许同一个数据记录拥有多个不同的版本。然后在查询时通过添加相对应的约束条件，就可以获取用户想要的对应版本的数据。

3.基本数据结构

（1）redo log：

重做日志记录。存储事务操作的最新数据记录，方便日后使用。

（2）undo log：

撤回日志记录,也称版本链。当前事务未提交之前，undo log保存了当前事务的正在操作的数据记录的所有版本的信息，undo log中的数据可作为数据旧版本快照供其他并发事务进行快照读。每次有其它事务提交对当前数据行的修改，都是添加到undo log中。undo log是由每个数据行的多个不同的版本链接在一起构成的一个记录“链表”。如下图：

事务A在操作DATA_ROW_ID=1的数据行的时候，事务B对这个数据行进行了更新操作，此时更新的信息会记录到这个undo log中