



NEW PLAN

软件高级架构师

• 一段新征程 •



- 整个直播课分为以下几种：
 - ✓ 知识点精讲直播：把一些考试出题频率特别高的章节进行重点、难点梳理+配套对应的真题讲解，以及一些记忆方法，建议同学们在直播前把对应的录播课程听完再来听直播，否则会感觉老师讲的快或者是听不懂的情况；
 - ✓ 案例分析的直播：讲解案例分析题型以及分析近10年来的案例分析出题规律，讲解下篇八大架构的案例分析重难点；
 - ✓ 论文专题的直播：从论文的如何书写到范文的讲解来进行剖析论文；
 - ✓ 历年真题的直播：把考试题目里把分值高的部分近5年来的所有题目拿出来进行一题一题的分析和讲解，分值高的知识点分别有：《数据库(选、案、论)：4-6分》、《软件工程+面向对象(选、案、论)：10-16分》、《架构设计+质量属性(选、案、论)：18-25分》、
- 直播时长一般会控制在2.5~3小时，根据不同的内容会有不同的时长表现，所以请同学们预留好时间；
- 直播之前，班主任老师会发放PDF课件，有时候PDF会有一些改动，如果改动大的话，会在直播之后重新发送给班主任老师，重新下载即可。



2026年5月份考试回顾-综合知识题



通过对综合知识题目的回顾，简单总结几点：

- 超纲题经过统计，有**14**道，达到了**20%**，仍然处于**20%**超纲率的范围内，超纲的题目五花八门，各种知识点的都有，跟去年下半年相比，符合老师预估的会出更多的大模型、人工智能之类的简单题目，除此之外，还出现了**2**道中级软设的题目（数据结构的最小生成树以及代码的编译过程），老师觉得这并不是一种固定的考法，随机性很强，就像是偶尔给你考几道比较深的网络题、嵌入式题或者是linux的题目，都算正常。
- 在之前的考试中，除了超纲题之外，还有一部分考的很细的，官方教材中的题目，本次经过老师简单统计，在选择题中暂时没发现这类题目，但是我觉得应该只是我没发现，不会没有，一般都会有**2-4**分的这种分值，各位同学还是要在完成了课程学习之后，正常去翻阅书籍。
- 除此之外，数据库中出现了几道不好拿分的题目，今年的数据库分值虽然比之前多了**1**分，但是有**2**道是超纲题，想拿满分并不容易
- 剩下的都是属于课程上讲过的内容，当然，也存在一些需要进行分析排除的题目，需要同学们在老师直播讲题中学会相关的分析和排除技巧才行。



2026年上选择题所涉及分值分布

章节	考分	考点	题目分布
1.计算机基础	5	3DES密钥长度、SSL/TSL属于哪种加密、数字签名、DDOS攻击方式、身份认证	1、28、47、48、62
2.操作系统知识	4	文件系统、父子进程、中断、存储空间分配方式	16、27、50、65
3.数据库	6	范式、自然连接、ER图中的联系（2分）、 实体与实体之间的联系（超纲）、倒排索引（超纲）	31、32、39、40、54、63
4.嵌入式技术	1	嵌入式系统设计思路	29
5.计算机网络	1	IEEE 802.11b传输速率	6
6.系统性能			
7.信息系统基础知识	1	专家系统	46
8.信息安全技术基础知识			
9.软件工程	8	边界值分析法（白盒测试）、V模型、测试左移、敏捷开发、冒烟测试、黑盒测试、结构化方法、软件复用	5、11、25、26、35、36、53、56
10.面向对象和结构化分析	3	装饰器设计模式、单例设计模式、UML用例图的关系	21、22、44
11.项目管理	1	工作量计算	4
12.系统架构设计	18	运行期质量属性、架构风格（5分）、ABSD（2分）、CS架构、DSSA（2分）、架构评估方法（3分）、质量属性（4）	2、10、12、13、17、18、41、42、43、49、51、55、57、58、61、68、69、70
13.软件可靠性基础知识			
14.软件架构的演化和维护			
15.未来信息综合技术	2	边缘计算、无监督学习	14、45
下篇八大架构	3	lambda（2分）、访问控制矩阵 ACM（安全架构）	7、8、9
数学与经济管理	2	函数的域、危险失效率	20、67
法律法规与标准化	3	专利（2分）、著作权	37、38、60
完全超纲	12	transformer、最小生成树（数据结构）、CQRS、DO-178B软件等级、编译过程、多模态、世界模型、知识图谱、AI数据投毒、微服务、知识库（题目出的很简单，送分）	3、15、19、23、24、30、33、34、52、59、64、66
书本上非常细节且上课完全没提过的知识			



2025年下选择题所涉及分值分布



知乎知学堂

章节	考分	考点	题目分布
1.计算机基础	4	基准测试、信息安全、PKI、电子签名	15、24、25、29
2.操作系统知识	4	文件管理、分时操作系统、信号量、强实时操作系统	12、14、38、66
3.数据库	4	关系代数、SQL、数据库设计、封锁协议	2、3、32、49
4.嵌入式技术	1	嵌入式系统的性能指标	54
5.计算机网络	2	子网掩码、ISO数据链路层	23、31
6.系统性能	1	性能设计	68
7.信息系统基础知识	1	专家系统	55
8.信息安全技术基础知识			
9.软件工程	10	逆向工程、测试用例、测试顺序、失效率、敏捷开发、内聚、软件复用、压力测试、软件生命周期划分原则、软件测试目的	8、13、16、17、35、44、49、61、67、69、70
10.面向对象和结构化分析	4	设计模式、数据流图、活动图、UML图	36、37、52、53
11.项目管理	2	工作量计算、最晚开工时间	10、45
12.系统架构设计	12	质量属性、权衡点、DSSA、MVP模型、ABSD、C/S架构、进程通信、规则系统（架构风格）、可移植性、CBAM、微服务	4、5、27、28、33、47、57、58、59、62、65
13.软件可靠性基础知识	1	可靠性测试	51
14.软件架构的演化和维护			
15.未来信息综合技术	1	监督学习	39
下篇八大架构	2	大数据架构、云原生	40、56
数学与经济管理	2	线性规划、容斥原理、排列组合	1、9
法律法规与标准化	2	商业秘密、专利	18、63
完全超纲	12	磁盘运算（操作系统）、机器学习X2、同步时钟（通信工程）、扇出扇入原则（程序设计原则）、嵌入式、ISO物理层、嵌入式数据流模型（程序设计）、工业产权（法律法规）、表示法、CMMI的知识体系（软件工程）、数据联邦（数据库）、EDI	11、19、20、21、22、26、30、34、41、46、60、64
书本上非常细节且上课完全没提过的知识	4	Hofmeister视图（架构）、头脑风暴质量属性（质量属性）、解释器（架构）、质量属性效用树（质量属性）	42、43、48、50
不好归类-难度低	2	更改加密算法属于什么分类、XML	6、7



2026年5月份考试回顾-案例分析题



- 第一题：必选，本次考试考了四大质量属性填空+质量属性六要素+架构风格+一道随机问答题（构件设计及职责），这题的前3问应该属于老师都命中的，质量属性的填空相比去年下半年是减弱的，这次就是ATAM的四个常见质量属性，第三问的回答你只要从过滤器的角度去回答几个不同功能的过滤器，都是可以拿到分值的；（难度比去年下半年降低，正常难度）
- 第二题：选做，是边缘协同的智能安防的考题，虽然看上去比较偏，但是问题都问的不难，比如第一问该系统的四层的主要职责，第二问则是选词填空，把具体的部件对应填入到这四层中，第三问是问这四层架构有哪些脆弱性缺点，整体来说每一问都可以回答一部分内容出来，难度合适；
- 第三题：选做，是一道数据库的Redis+秒杀场景的题，考察的是乐观锁、悲观锁以及高并发场景下的优化方式，悲观锁和高并发下的优化方式我们课程讲过，乐观锁则并未提及，还需要同学们自己平时积累相关知识；
- 第四题：选做，学习路径推荐系统，我拿到的题不全，有点像是一道偏前沿技术的题，不好做判断；
- 第五题：选做，偏嵌入式的老人居家健康监测系统，除了第一问选词填空简单点，后面两问对于学软件同学来说有些偏了；
- 总结：一般来说，5道题，第一道是必选的架构题，另外四道分别是数据库架构、web架构、嵌入式架构以及软件工程，也就是25年上半年考的很奇葩，除了数据库之外，另外3道都是很前沿知识点的题，现在都回归正常，也符合老师的判断后期不太可能出单独的软件工程的案例分析题了。

上一期案例分析押题

- ▶ 质量属性+架构风格+针对质量属性场景描述的提问+架构评估：必须熟悉运行期和开发期的所有质量属性，特别是几个容易混淆的（可用性、可靠性、鲁棒性、健壮性）+5大类架构风格，尽可能去看书本的原文和原图
- ▶ 架构填图题：一定的概率会从书本的下篇八大架构的案例中出架构图
- ▶ 数据库设计：Redis是绝对主角，特别是缓存与数据库的数据一致性、三大经典问题（穿透、击穿、雪崩）。并且大概率题目有高并发的业务背景（电商、物流、订购系统）以及这些背景下的一些经典解决方案（旁路缓存、布隆过滤器、互斥锁、主从复制与分片等），**一致性方面可能会考多级缓存一致性或分布式事务一致性，还有可能会涉及到集群分片（Redis Cluster）。**
- ▶ 软件工程：老师觉得现在单独出一道软件工程的案例分析题的概率已经不高了，只有可能把软件工程的知識作为一个子问题来进行出题，**大概率会跟架构填图题一起出。**
- ▶ 前沿技术：不好预测，按照现在的趋势，极有可能出NLP相关的，或者是人工智能的某个分支，这个题能不选就别选

2025年5月份考试回顾-论文题

- 向量数据库：第二问：让你回答向量数据库的原理和特点，并说明优点和不足
- 高并发系统：第二问：让你高并发系统设计的思路，可以从缓存设计、异步处理、限流降级、数据库优化、服务拆分、水平扩容等方面进行论述
- 多模态大模型：第二问：从框架的页面识别、规划测试路径、执行交互和分析几个层次进行论述，说明各层的作用及协同关系。
- 六边形架构设计方法：第二问：简述基于六边形架构思想的软件分析、设计和开发全流程，并说明核心概念及关键活动

总结：

1. 关于论文，老师一直以来的思路就是，从架构和软件工程这两个维度去准备题目，把所有对应的知识点都熟练掌握，题目如果完全命中最好，如果没有，那就找到最熟悉的地方往上面牵扯即可。
2. 从2025年开始，论文都是开始和实际工作相关的主题了，而不像往年，还有一些纯理论的话题（往年至少有一条道是理论类的话题），这就是一个趋势，并且人工智能的各个成熟的应用开始出现在论文里了。

论题一：云原生微服务架构与服务治理

预测理由：云原生是企业转型的绝对核心，但考题已从宽泛概念转向具体技术点。2025年5月考了事件驱动（微服务通信模式），2025年11月考了Serverless（微服务演进形态），那么微服务本身及其治理技术就是自然延伸。**服务网格、可观测性**等深化技术尚未独立命题，完全有可能考到。

涉及考察点：

- 微服务拆分原则（DDD领域驱动）
- 服务注册与发现（Eureka/Nacos）、配置中心（Apollo/Nacos）、API网关（Spring Cloud Gateway/Kong）
- 服务治理核心：熔断、降级、限流、负载均衡
- **可观测性体系：链路追踪（SkyWalking/Jaeger）、指标监控（Prometheus）、日志聚合（ELK/Loki）**
- **服务网格进阶：Sidecar模式（Envoy）、Istio流量管理、零信任安全（mTLS）**
- 容器化（Docker）与Kubernetes编排（Deployment/Service/Ingress/HPA自动扩缩容）

论题二：数据库架构与多模数据架构

预测理由：数据架构每年必考，2023-2025年已覆盖数据集成、Lambda架构、多模型数据库、云原生数据库。未来聚焦数据库技术深化与多模统一，向量数据库作为AI时代的新型数据存储，与传统关系/文档/图数据库共同构成多模架构。

可能考察点：

- 云原生关系数据库(TiDB/PolarDB/OceanBase)，一般应用在一些传统的领域，如交易、ERP、核心账务
- 图数据库(Neo4j/TuGraph)，一般应用在风控、社交关系、知识图谱等领域
- 向量数据库(Milvus/Weaviate/PGVector)，一般应用在RAG、推荐、相似度检索
- 实时采集：Kafka/Pulsar消息队列，CDC技术
- 流处理引擎：Flink窗口计算、状态管理 (Checkpoint)
- 实时OLAP：ClickHouse/Doris列式存储，预聚合优化

论题三：高并发系统性能优化场景

预测理由：考题明显转向场景解决方案。2024-2025年已考分布式事务、性能测试、秒杀场景，性能优化作为通用工程能力是自然延伸。

涉及考察点：

- 瓶颈分析：CPU/内存/IO/网络定位，火焰图分析，慢查询优化
- 多级缓存：本地缓存→分布式缓存（Redis）→CDN，缓存一致性
- 异步化：RocketMQ/Kafka削峰填谷，最终一致性，死信队列
- 系统韧性：熔断降级（Sentinel）、限流算法（令牌桶/漏桶）、舱壁隔离
- 容量保障：全链路压测、弹性伸缩、混沌工程

论题四：论AIGC技术对软件架构设计的影响与应用

预测理由：论文以后肯定会拿出一道题目来考未来技术，比如25年5月考了AI的其中一个应用方向（测试），那么下半年是否会考现在很火的大模型或者是某个具体的应用技术。

可能考察点：

- 如何利用AIGC进行设计文档辅助生成、技术方案辅助评审。
- 探讨AI编程助手对开发流程和架构模式的影响。
- 设计基于大模型的智能编程或系统运维助手。



2026年11月份考试备考计划



知乎知学堂

- 基础知识的学习（对应上午选择题）：全套视频的时长大概有55个小时，其中基础知识部分大概有45个小时，按照每周花5个小时来学习来算，我们需要9周完成学习，视频听完之后，用差不多45天的时间我们需要去把书本的软件工程基础知识（第五章）+架构设计基础知识（第七章）以及系统质量属性和架构评估（第八章）这三章给过一遍书，这三章总共加一起不到100页；
- 案例分析的学习（对应上午案例分析题）：案例分析的视频内容差不多有10个小时，听完之后，你就知道了案例分析的题型、考试方式以及答题技巧，然后通过我们专门的案例分析直播课+做真题的形式来完成学习即可，整个周期耗时差不多10天；
- 论文的学习（对应下午的论文题）：我们会在4月中旬这个时间开始讲论文课，分两次直播来进行讲解，讲解完成之后，需要你在4月剩下的时间完成2、3篇论文的书写，然后由老师进行批改。

注意：2026年6月的考试时间提前了半个月，要做好准备，10月24日至27日考试

学习方法总结

1. 给自己定好每天的学习时间，如果怕工作忙，就定每周的学习时间，务必把录播视频看完，看完之后可以利用碎片时间刷刷题；
2. 直播必须得听，因为直播除了串讲知识点之外，还会分享学习经验，考试趋势以及补充一些知识点；
3. 老师指定的书本上的那三章，必须要看书，不要抱有侥幸心理；
4. 案例分析考察知识面，除了第一题不超纲，后面4道题基本都会有部分的问题是超纲的，所以平时需要从各个方面积累下各个技术的解决经验，特别是数据库的，是比较好积累以及拿分的；
5. 论文必须要写，并且至少两篇，你听懂了和能写出来是两码事，直播会把论文书写讲的很清楚的，按照老师要求书写和送改；
6. 你不擅长的技术点，但是分数不高的，那么降低优先级，或者干脆放弃，集中精力放在分值高，容易拿分的知识点上；
7. 注意：如果直播和录播在某些知识点上有出入，以直播为准！





01

操作系统中几个 常考的知识点

- 本章节在改版前是有单独的一章的，改版之后，变成了一个小节，对应书本的**2.3.2**，篇幅很短，只有一些基本概念的介绍，但是老师还是按照老版本进行全面的录制课程，在改版后的近**4**次考试过程中，发现本章节仍然会以每次考试考**3-5**分的比例进行考察，所以同学们仍然还是需要对本章进行熟悉和掌握
- 之前被考到知识点有：
 - 进程管理：进程三态图、前趋图、同步与互斥、**PV**操作、死锁、线程。
 - 存储管理：分页存储管理、分段存储管理
 - 设备管理：IO软件层次
 - 文件管理：索引文件结构、文件目录、位示图计算
- 在改版之后的考试中，分别考察的知识点为：
 - **2023年11月**：多线程通信方式、单**CPU**任务交替运行、通道、内核功能、进程是资源分配最小单位
 - **2024年05月**：进程调度算法、多道程序设计、三态图、页表
 - **2024年11月**：死锁预防、三态图、段式存储、IO
 - **2025年05月**：页表和位示图、系统加载流程（超纲）、三态图、银行家算法、调度算法（超纲）
 - **2025年11月**：文件管理、分时操作系统、信号量、强实时操作系统、磁盘容量计算（超纲）
 - **2026年05月**：文件管理、父子进程（超纲）、中断、存储空间分配方式（更像是数据结构的题）

操作系统：进程管理-进程状态图

进程是计算机中正在运行的程序的实例。它是操作系统进行资源分配和管理的基本单位，包括代码、数据和执行状态等信息。

进程的组成：进程控制块PCB（唯一标志）、程序（描述进程要做什么）、数据（存放进程执行时所需数据）。

进程基础的状态是下左图中的三态图，这是系统自动控制时只有三种状态，而下右图中的五态，是多了两种状态：静止就绪和静止阻塞，需要人为的操作才会进入对应状态，活跃就绪即就绪，活跃阻塞即等待。



图 4-3 进程的三态模型

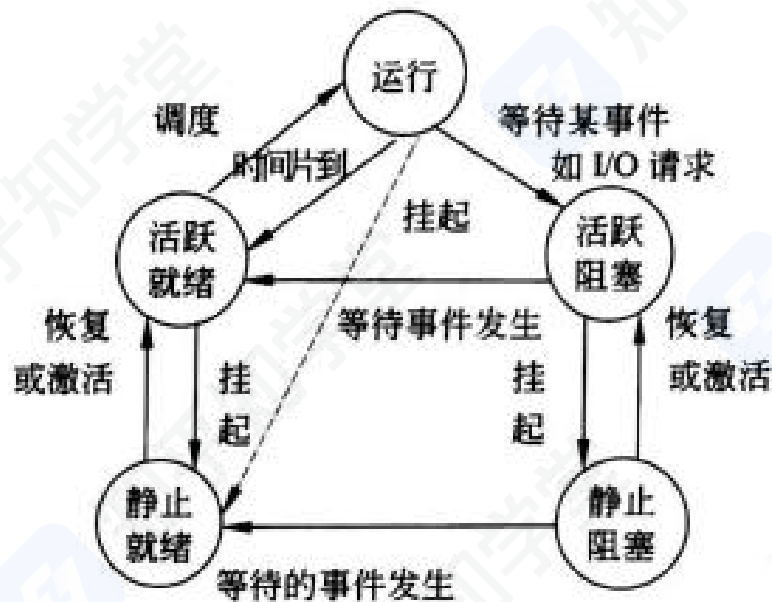
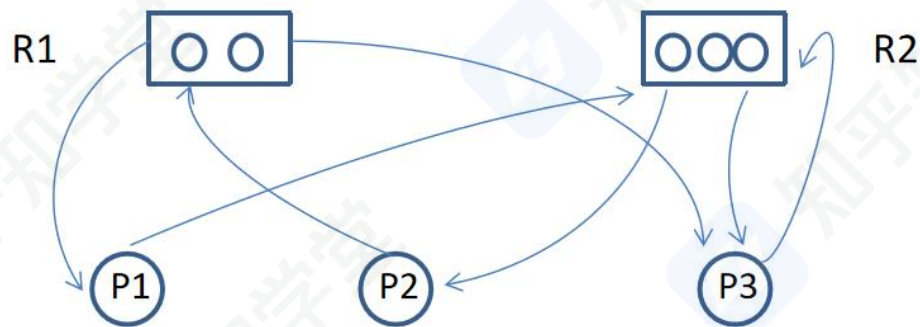


图 4-5 细分进程状态及其转换

操作系统：进程管理-进程资源图

进程资源图：用来表示进程和资源之间的分配和请求关系，如下图所示：



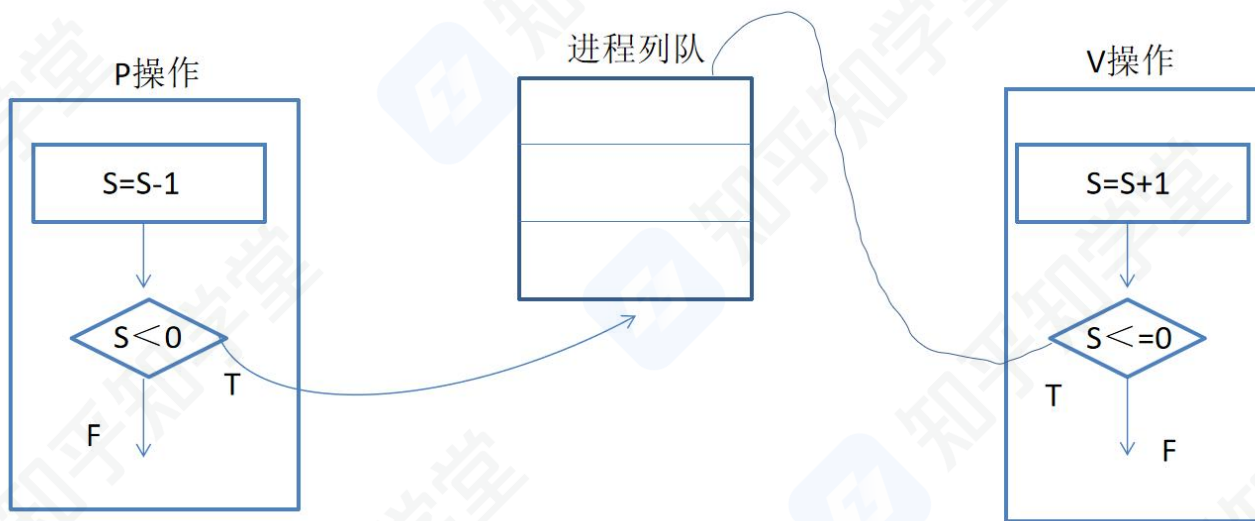
P代表进程，R代表资源，R方框中有几个圆球就表示有几个这种资源，在图中，R1指向P1，表示R1有一个资源已经分配给了P1，P1指向R2，表示P1还需要请求一个R2资源才能执行。

阻塞节点：某进程所请求的资源已经全部分配完毕，无法获取所需资源，该进程被阻塞了无法继续。如上图中P2。

非阻塞节点：某进程所请求的资源还有剩余，可以分配给该进程继续运行。如上图中P1、P3。
当一个进程资源图中所有进程都是阻塞节点时，即陷入死锁状态。

P操作：申请资源， $S=S-1$ ，若 $S \geq 0$ ，则执行P操作的进程继续执行；若 $S < 0$ ，则置该进程为阻塞状态（因为无可用资源），并将其插入阻塞队列。

V操作：释放资源， $S=S+1$ ，若 $S > 0$ ，代表此时资源有空余，没有阻塞的进程，则该进程继续执行；若 $S \leq 0$ ，代表此时线程在被阻塞，所以需要从阻塞状态唤醒一个进程，并将其插入就绪队列（此时因为缺少资源被P操作阻塞的进程可以继续执行），然后执行V操作的进程继续。



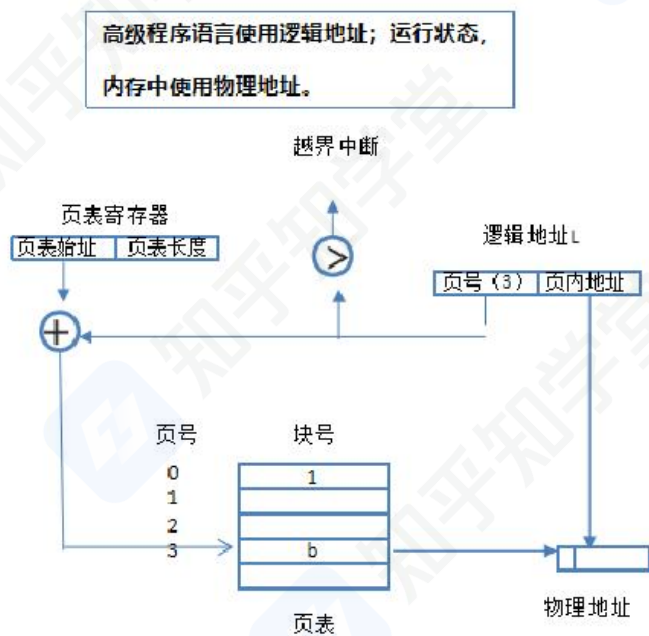
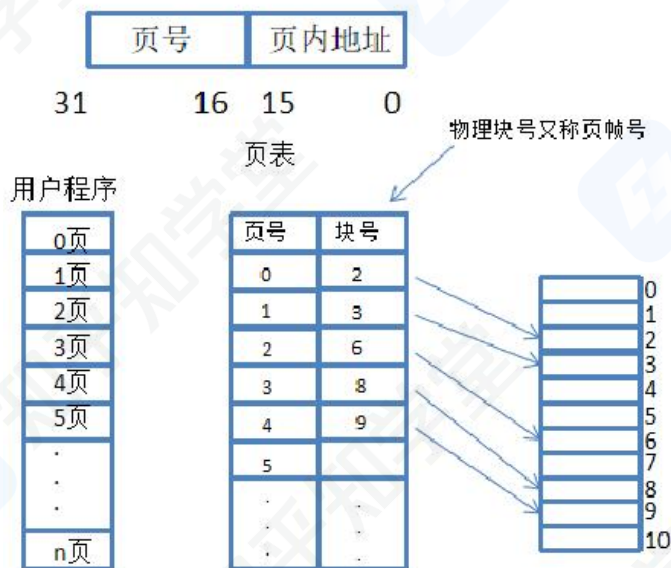
操作系统：存储管理-页式存储

页式存储是操作系统的一种存储管理方式。

因为我们的程序往往是远远大于内存的，所以程序在执行的时候，是不会一次性把所有内容都装入到内存中，它会把程序分为若干个**页**，每个页固定大小，一般是**4K**，然后把这些页离散存入到内存中，而**内存是按块来划分的**，所以就通过**页表**来进行映射程序中的页在内存中的块的存储；

进程(程序)中的地址，我们称之为**逻辑地址(虚地址)**，而内存中的地址我们称之为**物理地址(实地址)**；

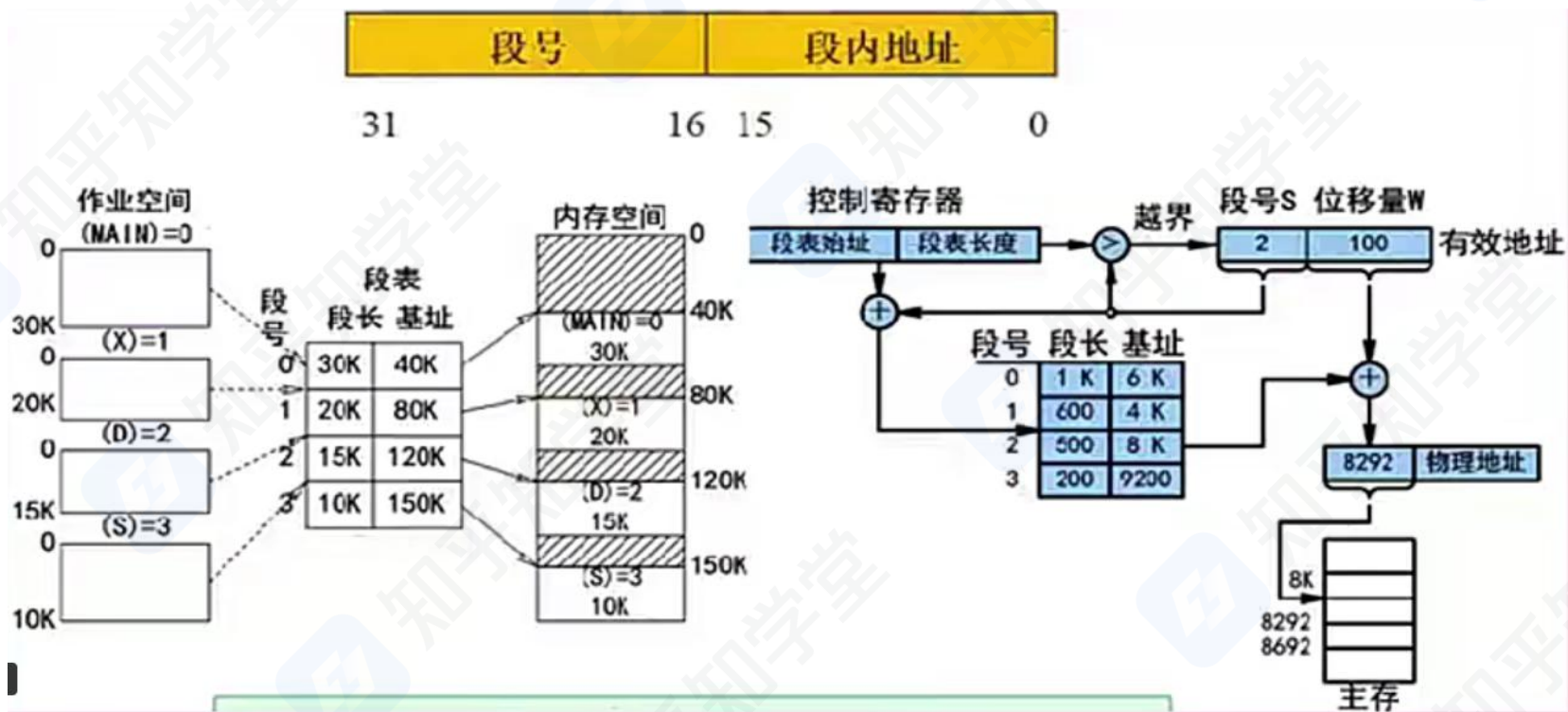
每个页分为**页号**和**页内地址**，**页号**用来和**块号**对应，代表存储的位置，大小可以代表页的数量，**页内地址**代表的是存储的数据内容，大小可以代表数据大小



操作系统：存储管理-段式存储

段式存储是指将进程空间分为一个个段，每段也有段号和段内地址，与页式存储不同的是，每段物理大小不同，分段是根据逻辑整体分段的。

地址表示：（段号，段内偏移）：其中段内偏移不能超过该段号对应的段长，否则越界错误，而此地址对应的真正内存地址应该是：段号对应的基地址+段内偏移。



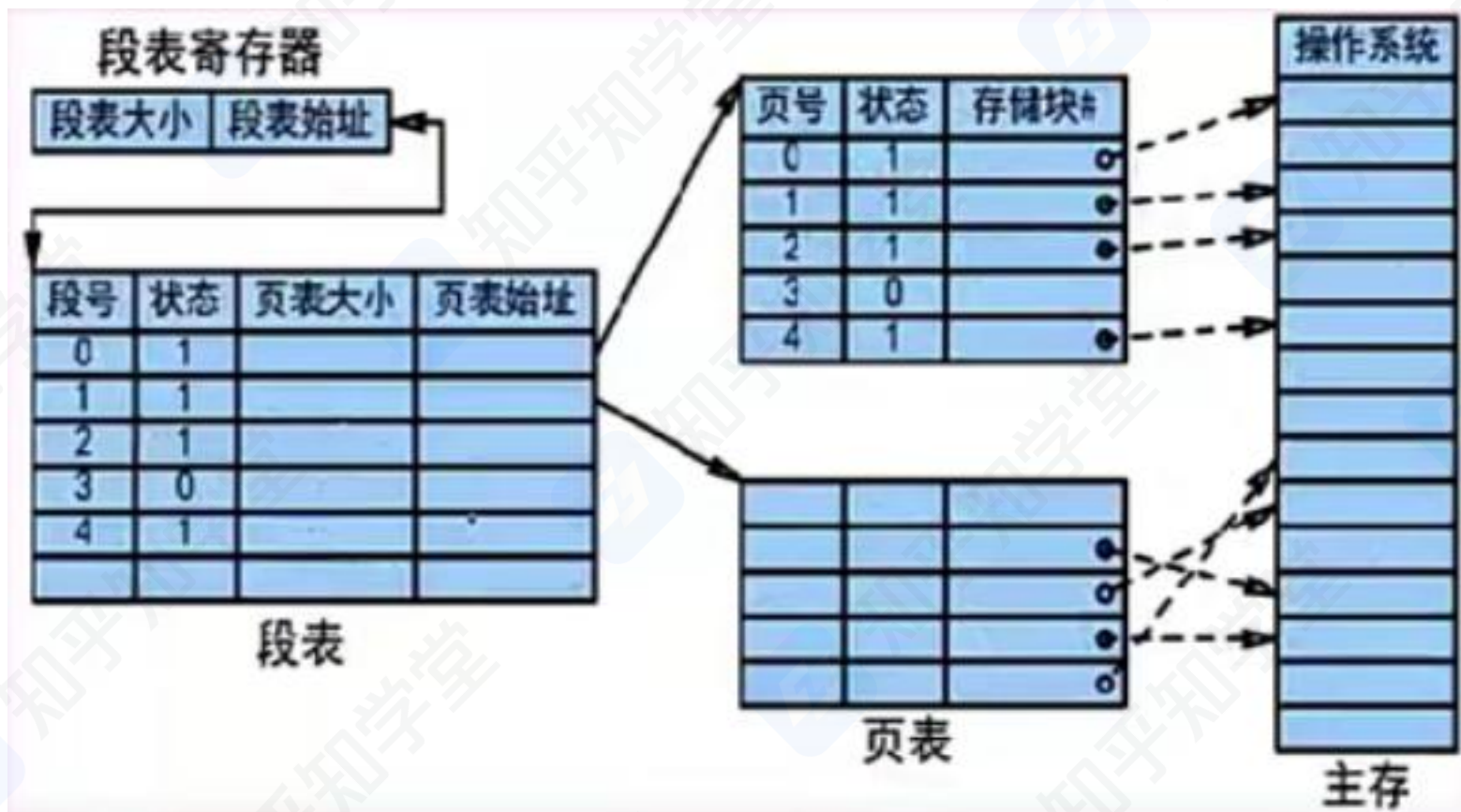
优点：程序逻辑完整，修改互不影响内存利用率低
缺点：内存碎片浪费大

操作系统-存储管理-段页式存储

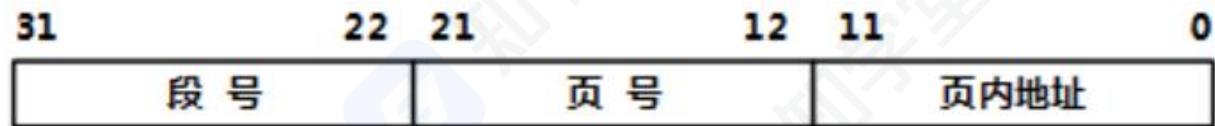
对进程空间先分段，后分页，具体原理图和优缺点如下：

优点：空间浪费小、存储共享容易、能动态连接。

缺点：由于管理软件的增加，复杂性和开销也增加，执行速度下降



例：假设段页式存储管理系统中的地址结构如下图所示，则系统（ ）。



- A.最多可有2048个段，每个段的大小均为2048个页，页的大小为2K
- B.最多可有2048个段，每个段最大允许有2048个页，页的大小为2K
- C.最多可有1024个段，每个段的大小均为1024个页，页的大小为4K
- D.最多可有1024个段，每个段最大允许有1024个页，页的大小为4K

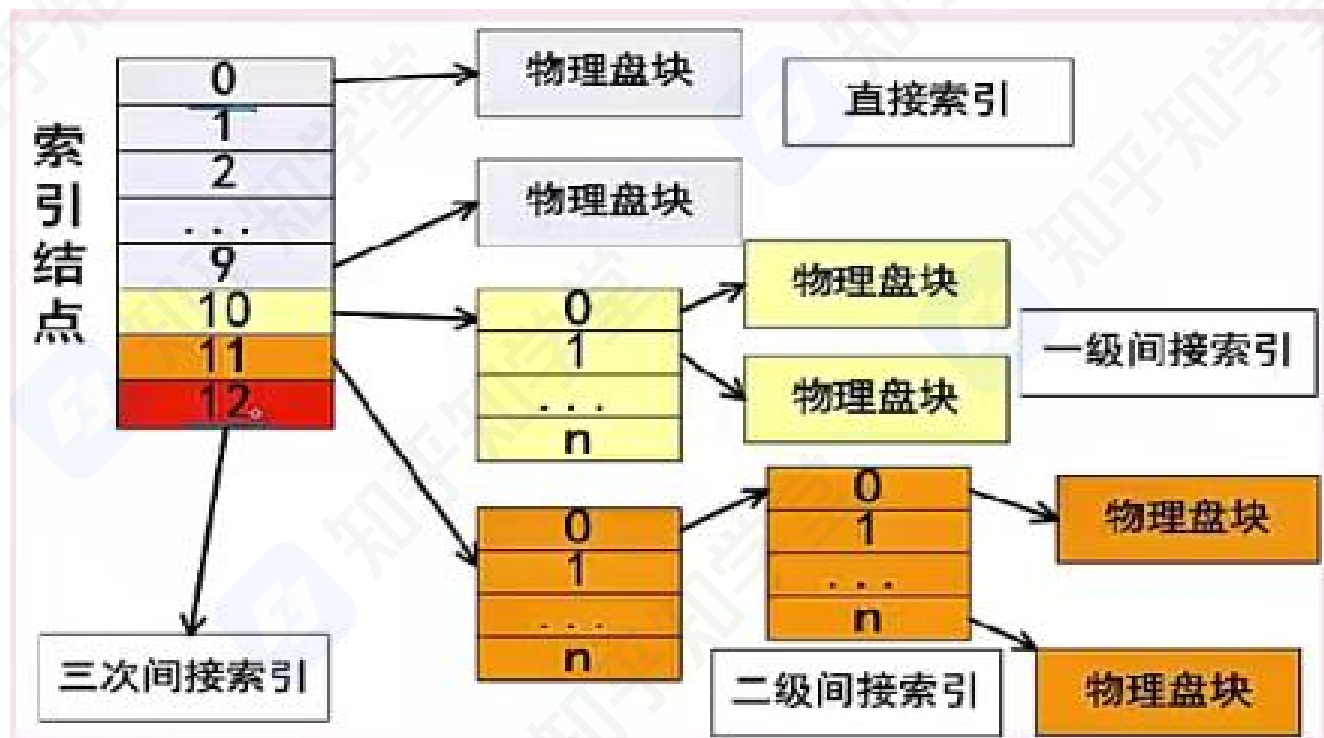
操作系统：文件管理-索引文件

计算机系统中采用的索引文件结构如下图所示：

系统中有13个索引节点，0-9为直接索引，即每个索引节点存放的是内容，假设每个物理盘大小为4KB，共可存 $4KB \times 10 = 40KB$ 数据；

10号索引节点为一级间接索引节点，大小为4KB，存放的并非直接数据，而是链接到直接物理盘块的地址，假设每个地址占4B，则共有1024个地址，对应1024个物理盘，可 $1024 \times 4KB = 4096KB$ 数据。

二级索引节点类似，直接盘存放一级地址，一级地址再存放物理盘快地址，而后链接到存放数据的物理盘块，容量又扩大了一个数量级，为 $1024 \times 1024 \times 4KB$ 数据。



磁盘空间存储的几种方式

- 空闲区表法：将所有空闲空间整合成一张表，即空闲文件目录。
- 空闲链表法：将所有空闲空间链接成一个链表，根据需要分配。
- 成组链接法：既分组，每组内又链接成链表，是上述两种方法的综合。
- 位示图法：对每个物理空间用一位标识，为1则使用，为0则空闲，形成一张位示图。

[illegible]

CPU读取设备的信息有以下几种常见的技术:

- 程序控制(查询)方式: CPU主动查询外设是否完成数据传输, 没传输完, CPU就一直等待, 效率极低。
- 通道技术 (中断技术): 目的是使数据的传输独立于CPU, CPU只需向通道发出IO命令, 通道收到命令后, 从主存中取出本次IO要执行的通道程序并执行, 完成后向 CPU 发出中断信号。缺点是一旦某通道被设备占用, 即使另一通道空闲, 连接该通道的其他设备也只有等待。适合于键盘鼠标等实时性较高的设备
- DMA技术 (直接主存存取): 是指数据在主存与IO设备间直接成块传送, 即在主存与I/O设备间传送一个数据块的过程中不需要CPU的任何干涉, 只需要CPU在过程开始启动与过程结束时的处理, 实际操作由DMA硬件直接执行完成, CPU 在此传送过程中可做别的事情。这种技术适合于硬盘等高速设备
- 缓冲技术: 缓冲技术可提高外设利用率, 尽可能使外设处于忙状态。缓冲技术可以采用硬件缓冲和软件缓冲。硬件缓冲是利用专门的硬件寄存器作为缓冲, 软件缓冲是通过操作系统来管理的。

Spooling技术: 通过磁盘缓冲区将打印机等独占设备虚拟为可共享的“多台虚拟设备”, 使多进程能并发使用, 提升设备利用率。



02

数据库



- 本章节在历年考试过程中的分值占比大概是**4-6分**，考试的内容在选择题方向的题目难度不大，偶尔会出现**1分**的超纲，在**2026年**上半年的考试过程中，总共出了**6分**，两分超纲，这也是近期数据库选择题少数超纲的考试，老师暂时无法判定是否会成为一种新的考试方式，我们还是正常准备即可
- 被考到知识点有：
 - 基本概念：三级模式-两级映像、数据库设计
 - 数据库模型：**E-R模型**、关系模型、关系代数、**SQL**
 - 规范化：函数依赖、键与约束、范式、模式分解
 - 事务并发：并发三种问题、三级封锁协议
 - 数据库新技术：数据库安全与备份、反规范化、分布式数据库、缓存数据库、数据库集群，**NoSql**
- 在改版之后的考试中，分别考察的知识点为：
 - **2023年11月**：范式（传递依赖和**2NF**,多值依赖和**4NF**)、数据库语句(**having+group by**)、三级模式
 - **2024年05月**：范式（**2NF**和部分依赖）、笛卡尔积、事务四大特性、关系代数换算、反规范化设计
 - **2024年11月**：约束、**SQL注入**、自然连接、函数依赖、三级模式、还有一道概念题
 - **2025年05月**：三级模式两级映像（考了**3题**）、关系代数、候选键
 - **2025年11月**：关系代数、**SQL**、数据库设计、封锁协议
 - **2026年05月**：三范式、自然连接、**ER图**中的联系（**2分**）、实体与实体之间的联系（超纲）、倒排索引（超纲）



数据库关系的三种类型：

- 基本表：实际存在的表，实际存储数据的逻辑表示。
- 查询表：查询结果对应的表
- 视图表：由基表或其他视图表导出的表，本身不独立存储，数据库只存放它的定义，常称为虚表。

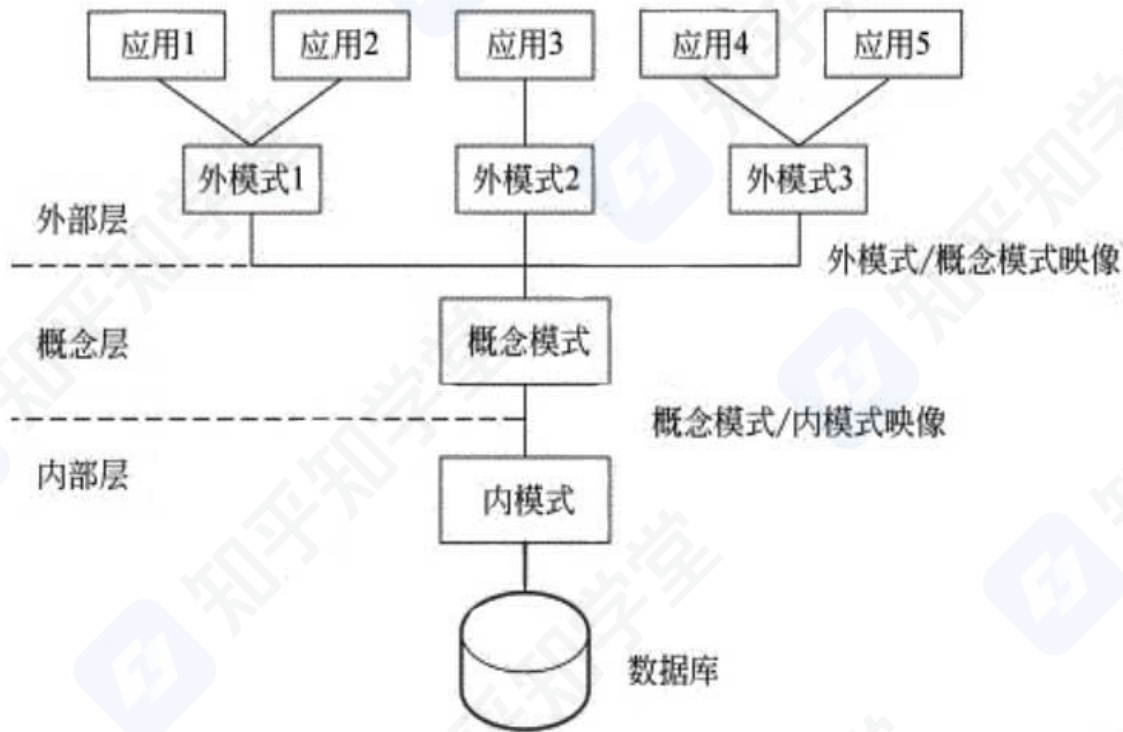
视图的优点：

- 视图能简化用户操作
- 视图使用户能以多种角度看待同一数据
- 视图对重构数据库提供了一定程度的逻辑独立性
- 视图可以对机密数据提供安全保护



三级模式是指数据库管理系统从三个层次来管理数据，分别是外部层（ExternalLevel）、概念层（Conceptual Level）和内部层（Internal Level）。这三个层次分别对应三种不同类型的模式，分别是外模式（External Schema）、概念模式（Conceptual Schema）和内模式（Internal Schema）。在外模式与概念模式之间，以及概念模式与内模式之间，还存在映像，即二级映像。

- 外模式：面向应用程序，描述用户的数据视图（View）；
- 内模式（又称为物理模式、存储模式）：面向物理上的数据库，描述数据在磁盘中如何存储；
- 概念模式（又称为模式、逻辑模式）：面向数据库设计人员，描述数据的整体逻辑结构。





50-51、数据库3级模式中，（ ）描述了记录的类型和记录间的关系，（ ）是用户需要使用的一部分数据的描述

。

50 A、概念模式

B、外模式

C、内模式

D、存储模式

51 A、概念模式

B、外模式

C、内模式

D、存储模式

66、数据库内模式描述的内容是？

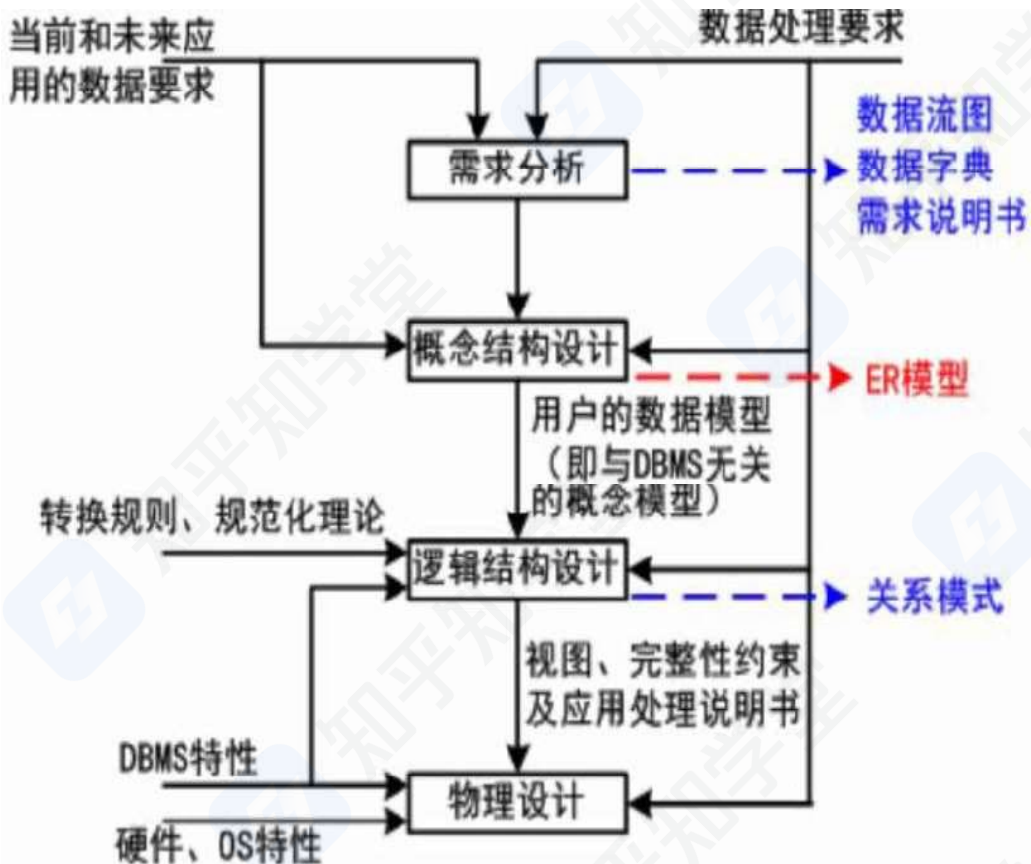
A. 存储结构

B. 索引

C. 物理存储

D. 数据类型

数据库系统-数据库设计



(1)需求分析：即分析数据存储的要求，产出物有数据流图、数据字典、需求说明书。获得用户对系统的三个要求：信息要求、处理要求、系统要求。

(2)概念结构设计：就是设计E-R图，也即实体-联系图。工作步骤包括：选择局部应用、逐一设计分E-R图、E-R图合并。分E-R图进行合并时，它们之间存在的冲突主要有以下3类。

- 属性冲突：属性自身规格不统一（类型、单位等），解决难度最低，统一规格即可。
比如：学号在不同部门定义不同，教务表：字符型(2023001)；学生表：整数型(2023001)
- 命名冲突：名称与含义不匹配（同名异义 / 异名同义），解决难度较低，统一命名规范即可。比如：老师在人事表称"在编职工"，在教务表中称"教师"，在财务表中称"员工"
- 结构冲突：实体 / 联系的抽象结构不一致，解决难度最高，需要协商统一抽象标准。
比如：员工实体属性不一致，人事E-R图含(工号、名、性、生日、部门)；教学E-R图含(教工号、名、职位、专业、学位)，需取并集

(3)逻辑结构设计：将E-R图，转换成关系模式。工作步骤包括：确定数据模型、将E-R图转换成为指定的数据模型、确定完整性约束和确定用户视图。

(4)物理设计：步骤包括确定数据分布、存储结构和访问方式。

(5)数据库实施阶段：根据逻辑设计和物理设计阶段的结果建立数据库，编制与调试应用程序，组织数据入库，并进行试运行。

(6)数据库运行和维护阶段：数据库应用系统经过试运行即可投入运行，但该阶段需要不断地对系统进行评价、调整与修改。

冲突类型	核心定义	具体表现分类	示例	核心问题	解决思路
属性冲突	同一属性在不同E-R图中，规格（数据类型、精度、单位、取值范围等）不一致。	① 属性域冲突：数据类型、精度或取值范围不同。 ② 属性取值单位冲突：度量单位不同。	- 「员工工资」：财务为 decimal(10,2)，人事为 int。 - 「员工身高」：人事用厘米，体检中心用米。	属性的“规格”不统一。	统一属性定义：确定标准的数据类型、精度、取值范围和单位。
命名冲突	名称与所代表的意义不匹配，出现命名重复或不对应的情况。	① 同名异义：名称相同，含义不同。 ② 异名同义：名称不同，含义相同。	- 「客户」：销售是“购买者”，财务是“欠款者”。 - 「员工工号」「薪资编号」「考勤编号」都代表同一员工编号。	名称与概念对应混乱。	统一命名规范或建立名称映射表，确保“同名同义、异名区分”。
结构冲突	同一对象在不同E-R图中，抽象方式或结构定义不同（如实体、属性或联系定义不一致）。	① 实体属性差异：同一实体的属性集合不同。 ② 抽象层次不同：同一对象被定义为实体或属性。 ③ 联系类型不同：联系的基数（1:1、1:n、m:n）不一致。	- 「学生」实体在不同部门属性不同。 - 「课程成绩」在教务是实体，在财务是属性。 - 「教师-课程」联系在小学为1:n，大学为m:n。	对事物的抽象方式不一致。	各部门协商统一抽象标准，统一实体、联系、属性的定义。



数据库系统-冲突相关练习



1.某高校信息系统设计的分E-R图中,
人力部门定义的职工实体具有属性: 职工号、姓名、性别和出生日期;
教学部门定义的教师实体具有属性: 教师号、姓名和职称。
这种情况属于(), 在合并E-R图时, ()解决这一冲突。

- A.属性冲突 B.命名冲突 C.结构冲突 D.实体冲突

- A.职工和教师实体保持各自属性不变
B.职工实体中加入职称属性, 删除教师实体
C.教师也是学校的职工, 故直接将教师实体删除
D.将教师实体所有属性并入职工实体, 删除教师实体

2.在某企业的营销管理系统设计阶段, 属性“员工”在考勤管理子系统中被称为“员工”,
而在档案管理子系统中被称为“职工”, 这类冲突称为()冲突。

- A.语义 B.结构 C.属性 D.命名

用E-R图来描述概念数据模型，世界是由一组称作实体的基本对象和这些对象之间的联系构成的。

在E-R模型中，使用椭圆表示属性(一般没有)、长方形表示实体、菱形表示联系，联系的两端要填写联系类型，示例如下图：

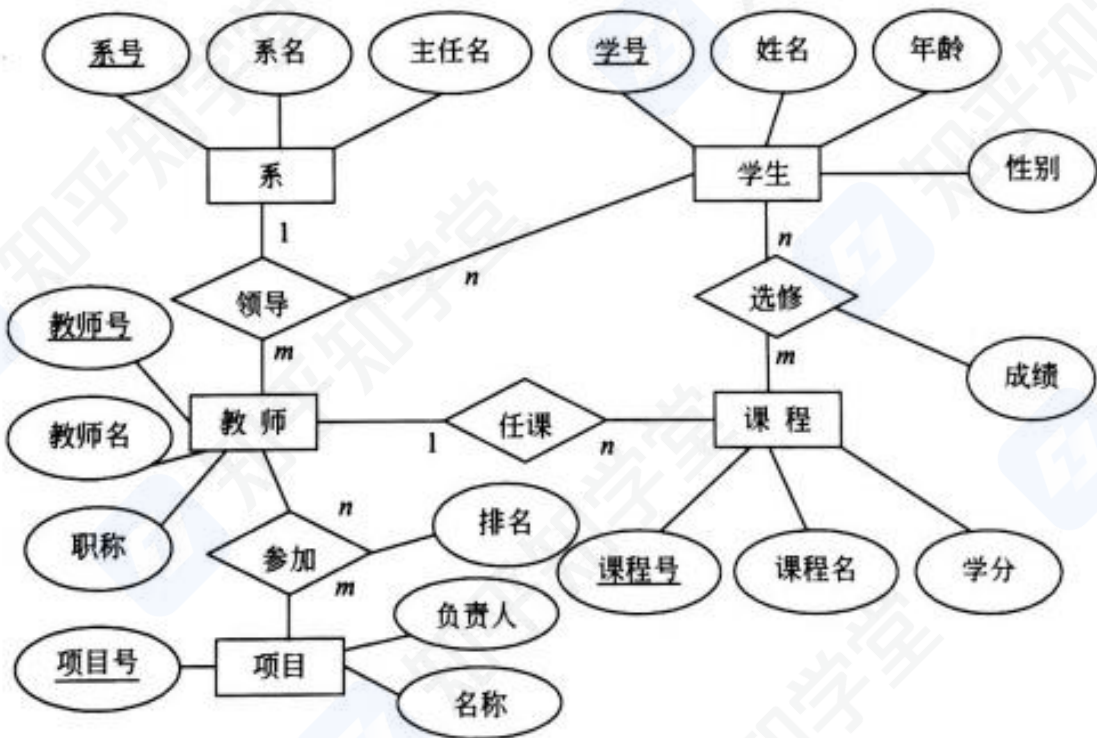


图 9-9 学校教学管理系统的 E-R 模型

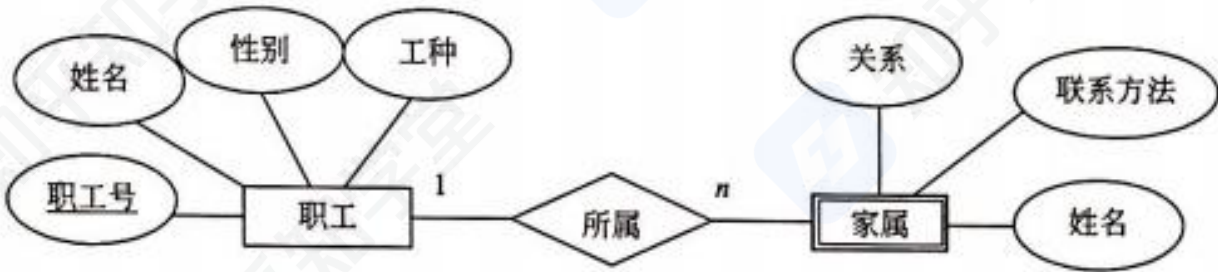


图 9-10 弱实体与依赖联系

关系模型中数据的逻辑结构是一张二维表，由行列组成。用表格结构表达实体集，用外键标识实体间的联系。

- 优点：建立在严格的数学概念基础上；概念单一、结构简单、清晰，用户易懂易用；存取路径对用户透明，从而数据独立性、安全性好，简化数据库开发工作。
- 缺点：由于存取路径透明，查询效率往往不如非关系数据模型。

S 学生关系					T 教师关系			
Sno	Sname	SD	Age	Sex	Tno	Tname	Age	Sex
01001	贾皓昕	IS	20	男	001	方铭	34	女
01002	姚勇	IS	20	男	002	章雨敬	58	男
03001	李晓红	CS	19	女	003	王平	48	女

SC 选课			C 课程关系		
Sno	Cno	Grade	Cno	Cname	Pcno
01001	C001	90	C001	MS	
01001	C002	91	C002	IC	
01002	C001	95	C003	C++	C002
01002	C003	89	C004	OS	C002
03001	C001	91	C005	DBMS	C004

图 9-12 关系模型的实例

表 9-2 关系代数运算符

运 算 符		含 义	运 算 符		含 义
集合运算符	$\cup$	并 差 交 笛卡儿积	比较运算符	$>$	大于
	$-$			$\geq$	大于等于
	$\cap$			$<$	小于
	$\times$			$\leq$	小于等于
				$=$	等于
				$\neq$	不等于
专门的关系运算符	σ	选择 投影 连接 除	逻辑运算符	$\neg$	非 与 或
	π			$\wedge$	
	$\bowtie$			$\vee$	
	$\div$				

【例 9.4】 设有关系 R 、 S 如下所示，请求出 $R \cup S$ 、 $R - S$ 、 $R \times S$ 、 $\pi_{A,C}(R)$ 、 $\sigma_{A>B}(R)$ 和 $\sigma_{3<4}(R \times S)$ 。

A	B	C
a	b	c
b	a	d
c	d	e
d	f	g

关系 R

A	B	C
b	a	d
d	f	g
f	h	k

关系 S

R ∪ S		
A	B	C
a	b	c
b	a	d
c	d	e
d	f	g
f	h	k

R - S		
A	B	C
a	b	c
c	d	e

$\pi_{A,C}(R)$	
A	C
a	c
b	d
c	e
d	g

$\sigma_{A>B}(R)$		
A	B	C
b	a	D

R × S					
R.A	R.B	R.C	S.A	S.B	S.C
a	b	c	b	a	d
a	b	c	d	f	g
a	b	c	f	h	k
b	a	d	b	a	d
b	a	d	d	f	g
b	a	d	f	h	k
c	d	e	b	a	d
c	d	e	d	f	g
c	d	e	f	h	k
d	f	g	b	a	d
d	f	g	d	f	g
d	f	g	f	h	k

$\sigma_{3<4}(R \times S)$					
R.A	R.B	R.C	S.A	S.B	S.C
a	b	c	d	f	g
a	b	c	f	h	k
b	a	d	f	h	k
c	d	e	f	h	k

图 9-15 运算结果



给定关系 $R(A, B, C, D)$ 和关系 $S(C, D, E)$ ，对其进行自然连接运算 $R \bowtie S$ 后的属性列为 () 个；

与 $\sigma_{R.B > S.E} (R \bowtie S)$ 等价的关系代数表达式为 ()。

A. 4

B. 5

C. 6

D. 7

A. $\sigma_{2>7}(R \times S)$

B. $\pi_{1,2,3,4,7}(\sigma_{'2' > '7' \wedge 3=5 \wedge 4=6}(R \times S)) \downarrow$

C. $\sigma_{'2' > '7'}(R \times S)$

D. $\pi_{1,2,3,4,7}(\sigma_{2>7 \wedge 3=5 \wedge 4=6}(R \times S)) \downarrow$

【2025年上】6、关系代数运算不包括()

- A、选择
- B、投影
- C、删除
- D、连接

【2025年下】2.数据库R- (R-S) 与 () 等价。

- A. $R \cap S$
- B. $R \cup S$
- C. $R - S$
- D. $R * S$

【2025年下】3.在Sql中，select执行结果是 ()

- A.元组
- B.集合
- C.属性值
- D.序偶



【2026年上】关系 $R(A,B)$ 有 5 个元组，且 B 是 R 的主键， $S(B,C)$ 有 10 个元组，则 R 与 S 按属性 B 进行自然连接后，结果元组数的取值范围是 ()。

- A、 $[0,5]$
- B、 $[0,10]$
- C、 $[5,10]$
- D、 $[0,50]$

【2026年上】某业务规则规定一个学生必须且只能属于一个班级；每个班级至少 10 人，至多 50 人。则学生实体参与该联系的基数应表示为 ()。

- A、 $(10,50)$
- B、 $(0,10)$
- C、 $(0,50)$
- D、 $(1,1)$



【2026年上】在关系数据库中，实体与实体之间的联系通常通过表与表之间的（ ）来体现。

- A、公共属性
- B、顺序号
- C、页号
- D、聚簇索引名

【2026年上】倒排索引最主要的作用是（ ）。

- A、加速文档内容搜索
- B、优化数据的存储结构，降低存储开销
- C、加快数据新增与修改的执行速度
- D、梳理数据关联，简化多表查询逻辑



函数依赖：给定一个X，能唯一确定一个Y，就称X决定(确定)Y，或者说Y依赖于X。

- 例如： $Y=X*X$ 函数，此时X能确定Y的值，但是Y无法确定X的值，比如 $x=2$ ， $y=4$ ，但是 $y=4$ 无法确定 $x=2$ 。

函数依赖又可扩展以下两种规则：

- 部分函数依赖：部分函数依赖是指在一个关系模式中，一个非主属性（即非码属性）依赖于关系模式中的某个码的一部分，而不是整个码。
- 传递函数依赖：指的是当一个非主属性依赖于另一个非主属性，而这两个非主属性之间又通过主属性之间的依赖关系联系起来时，就存在传递依赖。

部分函数依赖举例：假设有一个关系模式 $R = \{A, B, C\}$ ，其中 $\{A, B\}$ 是 R 的码，如果某个非码属性（比如 C）只依赖于 R 的码中的一部分（比如 A），而不是整个码（即 $\{A, B\}$ ），那么我们说 C 对 A 部分函数依赖（记为 $A \twoheadrightarrow C$ ）。

传递函数依赖举例：假设有一个订单表（Orders），包含字段 {订单号, 客户ID, 客户姓名, 订单日期}，并且存在以下函数依赖：订单号 $\rightarrow$ 客户ID、客户ID $\rightarrow$ 客户姓名、订单号 $\rightarrow$ 订单日期，在这个例子中，“订单号”是“客户ID”和“订单日期”的候选键。存在传递函数依赖：订单号 $\rightarrow$ 客户ID $\rightarrow$ 客户姓名。这意味着如果我们知道订单号，我们可以确定客户ID，进而确定客户姓名。

第一范式 (1NF)： 若关系模式R的每一个分量是不可再分的数据项，则关系模式R属于第一范式。通俗地说，第一范式就是表中不允许有小表的存在。比如，对于如下的员工表，就不属于第一范式：

员工编号	员工姓名	出生日期	薪资/月		所属部门
			基本工资/月	补贴/月	
1	王红	19900908	9000	1000	101
...	...	...	...	...	...

例如，供应者和它所提供的零件信息，关系模式FIRST和函数依赖集F如下：

- FIRST(Sno,Sname,Status,City,Pno,Qty)
- $F=\{Sno \rightarrow Sname, Sno \rightarrow Status, Status \rightarrow City, (Sno, Pno) \rightarrow Qty\}$

表 9-8 FIRST

Sno	Sname	Status	City	Pno	Qty
S1	精 益	20	天津	P1	200
S1	精 益	20	天津	P2	300
S1	精 益	20	天津	P3	480
S2	盛 锡	10	北京	P2	168
S2	盛 锡	10	北京	P3	500
S3	东方红	30	北京	P1	300
S3	东方红	30	北京	P2	280
S4	泰 达	40	上海	P2	460

第二范式 (2NF)：在1NF的基础上，要求数据库表中的每个非主属性完全依赖于码，例如，FIRST关系中的码是Sno、Pno，而Sno→Status,因此非主属性Status部分函数依赖于码，故非2NF的。

表 9-8 FIRST

Sno	Sname	Status	City	Pno	Qty
S1	精 益	20	天津	P1	200
S1	精 益	20	天津	P2	300
S1	精 益	20	天津	P3	480
S2	盛 锡	10	北京	P2	168
S2	盛 锡	10	北京	P3	500
S3	东方红	30	北京	P1	300
S3	东方红	30	北京	P2	280
S4	泰 达	40	上海	P2	460

若此时将FIRST关系分解为FIRST1(Sno,Sname,Status,City)和FIRST2(Sno,Pno, Qty)。分解后的关系模式FIRST1的码为Sno，非主属性Sname、Status、City完全依赖于码Sno,所以属于2NF； 关系模式FIRST2的码为Sno、Pno,非主属性Qty完全依赖于码，所以也属于2NF。

第三范式 (3NF)： 在2NF的基础上，消除了非主属性对码的传递函数依赖，则称为3NF。例如，FIRST1不属于3NF,因为在分解后的关系模式FIRST1中有Sno→Status, Status→City，存在着非主属性City传递依赖于码Sno。若此时将FIRST1继续分解为： FIRST1.1 (Sno,Sname,Status)、 FIRST1.2 (Status,City)，通过上述分解，数据库模式FIRST转换为 FIRST1.1(Sno,Sname,Status)、 FIRST1.2(Status,City)和FIRST2(Sno,Pno,Qty)3个子模式。由于这3个子模式都达到了3NF，因此称分解后的数据库模式达到了3NF。

可以证明，3NF的模式必是2NF的模式。产生冗余和异常的两个重要原因是部分依赖和传递依赖。因为3NF模式中不存在非主属性对码的部分函数依赖和传递函数依赖，所以具有较好的性能。对于非3NF的1NF、2NF其性能弱，一般不宜作为数据库模式，通常要将它们变换成为3NF或更高级别的范式，这个变换过程称为“关系模式的规范化处理”。

第四范式：在3NF的基础上，要求一个表的主键只对应一个多值。
例如，学生信息表(学生ID, 住址, 电话号码)，这个表中住址和电话号码表示学生可以有多个，它们与学生存在多值依赖关系。例子里的学生信息表同时包含“多个住址”和“多个电话”两个独立多值，因此不满足 4NF。因为住址和电话号码是独立的多值依赖，这意味着它们各自都直接依赖于学生ID，并且彼此之间是独立的。解决办法有两种，一种是通过程序来控制，二种是拆成两张表，每张表里面只拥有一个多值。

学生 ID (主键)	住址 (多值)	电话号码 (多值)
2023001	北京市朝阳区 XX 小区 1 号楼	13800138001 (父亲)
2023001	北京市朝阳区 XX 小区 1 号楼	13900139001 (本人)
2023001	北京市海淀区 XX 学校宿舍 3 栋	13800138001 (父亲)
2023001	北京市海淀区 XX 学校宿舍 3 栋	13900139001 (本人)
2023002	上海市浦东新区 XX 公寓 5 号楼	13700137001 (母亲)

BC范式(BCNF): 也称之为三范式增强, 在符合第三范式的基础上, 要求关系中所有非平凡函数依赖 (即若存在 $X \rightarrow Y$, 且 Y 不包含于 X) 的“决定因素 X ”必须是该关系的候选键 (能唯一确定一行数据的最小属性集)。

简单来说, 就是不允许出现“联合主键的子集 $\rightarrow$ 主属性 / 非主属性”的依赖

假设仓库管理关系表(仓库, 管理员, 物品名, 数量), 且每个仓库只能有一名管理员, 一名管理员只能在一个仓库中工作, 一个仓库中可以存放多种物品, 一种物品也可以存放在不同的仓库中。每种物品在每个仓库中都有对应的数量, 如右图所示:

此关系模式已经属于了3NF, 那么这个关系模式是否存在问题呢? 我们来看以下几种操作:

- 先新增加一个仓库, 但尚未存放任何物品, 是否可以为该仓库指派管理员? ——不可以, 因为物品名也是主属性, 根据实体完整性的要求, 主属性不能为空。
- 某仓库被清空后, 需要删除所有与这个仓库相关的物品存放记录, 会带来什么问题? ——仓库本身与管理员的信息也被随之删除了。
- 如果某仓库更换了管理员, 会带来什么问题? ——这个仓库有几条物品存放记录, 就要修改多少次管理员信息。
- 从这里我们可以得出结论, 在某些特殊情况下, 即使关系模式符合 3NF 的要求, 仍然存在着插入异常, 修改异常与删除异常的问题, 仍然不是 “好” 的设计。

解决方案: 把仓库管理关系表分解为二个关系表:

- 仓库管理: (仓库名, 管理员)
- 仓库库存: (仓库名, 物品名, 数量)

仓库名	管理员	物品名	数量
上海仓	张三	iPhone 5s	30
上海仓	张三	iPad Air	40
北京仓	李四	iPhone 5s	50
北京仓	李四	iPad Mini	60

候选键: (管理员, 物品名), (仓库名, 物品名)

主属性: 仓库名、管理员、物品名,

非主属性: 数量

函数依赖集: 仓库名 $\rightarrow$ 管理员, 管理员 $\rightarrow$ 仓库名, (仓库名/管理员, 物品名) $\rightarrow$ 数量

是否存在部分和传递函数依赖?

是否出现联合主键的子集能够决定主属性的情况?



2026年上：综合知识



知乎知学堂

【2026年上】关系模式 $R(A,B,C)$ 中，若存在函数依赖 $A \twoheadrightarrow B$ ， $B \twoheadrightarrow C$ ，则该关系模式最高满足()

- A.1NF
- B.2NF
- C.3NF
- D.BCNF



函数依赖的公理系统(Armstrong阿姆斯特朗公理)是一组在关系数据库理论中用于推导属性依赖的基本规则。提供了一种形式化的方法，用于推导关系数据库中的所有属性依赖。通过使用这套公理，可以理解和掌握一个数据库的所有潜在的属性依赖，从而帮助设计合理的数据库模式，确保数据的一致性与完整性。

Armstrong公理包括三条基本规则，它们分别是：

- 自反律：若属性集 Y 是属性集 X 的子集 ($Y \subseteq X \subseteq U$)，则 $X \rightarrow Y$ 成立。这表示如果一个属性集合 Y 是另一个属性集合 X 的子集，那么 X 可以决定 Y 。
- 增广律：若 $X \rightarrow Y$ 成立，且属性集 Z 是属性集 U 的子集 ($Z \subseteq U$)，则 $XZ \rightarrow YZ$ 也成立。这表示如果 X 可以决定 Y ，那么在任意集合 Z 加入到 X 和 Y 两侧的情况下，依赖关系仍然成立。
- 传递律：若 $X \rightarrow Y$ 及 $Y \rightarrow Z$ 成立，则 $X \rightarrow Z$ 也成立。这表示如果 X 可以决定 Y ，且 Y 可以决定 Z ，那么 X 也可以决定 Z 。

根据上述3条推理规则又可推出下述3条推理规则：

- 合并律：对于任意属性集合 X 、 Y 和 Z ，如果 $X \rightarrow Y$ 和 $X \rightarrow Z$ ，那么 $X \rightarrow YZ$ 。
- 分解律：对于任意属性集合 X 、 Y 和 Z ，如果 $X \rightarrow YZ$ ，则 $X \rightarrow Y$ 和 $X \rightarrow Z$ 。
- 伪传递规则：若 $X \rightarrow Y$ 成立，且 $WY \rightarrow Z$ 成立，则 $XW \rightarrow Z$ 也成立。这表示如果 X 可以决定 Y ，且 WY 可以决定 Z ，那么 XW 也可以决定 Z 。这条规则适用于处理那些不仅依赖于单个属性集合的复杂函数依赖关系。



模式分解：是关系数据库规范化设计中的一个重要概念。它指的是通过对关系模式进行拆分，来消除模式中的混合组合依赖，达到将模式分解为更小的模式的过程。一般分为以下两种：

- 是否保持函数依赖分解：对于关系模式R，有依赖集F，若对R进行分解，分解出来的多个关系模式，保持原来的依赖集不变，则为保持函数依赖的分解。另外，注意要消除掉冗余依赖(如传递依赖)
- 有损无损分解：分解后的关系模式能够还原出原关系模式，就是无损分解，不能还原就是有损

是否保持函数依赖：设原关系模式R(A, B, C)，依赖集F(A→B, B→C, A→C)，将其分解为两个关系模式R1(A, B)和R2(B, C)，此时R1中保持依赖A→B，R2保持依赖B→C，说明分解后的R1和R2是保持函数依赖的分解，因为A→C这个函数依赖实际是一个冗余依赖，可以由前两个依赖传递得到，因此不需要管。

是否无损分解：如果R的分解为 $p=\{R_1, R_2\}$ ，F为R所满足的函数依赖集合，分解p具有无损连接性的充分必要条件是 $R_1 \cap R_2 \rightarrow (R_1 - R_2)$ 或者 $R_1 \cap R_2 \rightarrow (R_2 - R_1)$ 。

假设关系模式 $R(U, F)$ ，属性集 $U=\{A, B, C\}$ ，函数依赖集 $F=\{A \rightarrow B, B \rightarrow C\}$ 。若将其分解为 $p=\{R_1(U_1, F_1), R_2(U_2, F_2)\}$ ，其中 $U_1=\{A, B\}$ ， $U_2=\{A, C\}$ 。那么，分解 p ()。

- A.有损连接但保持函数依赖
- B.既无损连接又保持函数依赖
- C.有损连接且不保持函数依赖
- D.无损连接但不保持函数依赖

给定关系模式 $R\langle U, F \rangle$ ， $U=\{A, B, C, D, E\}$ ， $F=\{B \rightarrow A, D \rightarrow A, A \rightarrow E, AC \rightarrow B\}$ ，则 R 的候选关键字为()，分解 $p=\{R_1(ABCE), R_2(CD)\}$ ()。

- A.CD
- B.ABD
- C.ACD
- D.ADE

- A.具有无损连接性，且保持函数依赖
- B.不具有无损连接性，但保持函数依赖
- C.具有无损连接性，但不保持函数依赖
- D.不具有无损连接性，也不保持函数依赖



2025年上：综合知识



知乎知学堂

41、数据库候选键， $R(a,b,c,d)$ ， $a \rightarrow cd$ ， $c \rightarrow b$ ，候选码是（）

A.a

B.b

C.c

D.d



事务：由一系列DML操作组成，这些操作，要么全做，要么全不做，它从第一个DML操作开始，rollback、commit或者DDL结束，拥有以下四种特性，详解如下：

- (操作)原子性：要么全做，要么全不做，例如银行转账，在没到最后一步完成转账之前，前面的所有操作都是无效的。
- (数据)一致性：事务发生后数据是一致的，例如银行转账，不会存在A账户转出，但是B账户没收到的情况。
- (执行)隔离性：任一事务的更新操作直到其成功提交的整个过程对其他事务都是不可见的，不同事务之间是隔离的，互不干涉。
- (改变)持续性：事务操作的结果是持续性的。

事务是并发控制的前提条件，并发控制就是控制不同的事务并发执行，提高系统效率，但是并发控制中存在下面三个问题：

丢失更新：事务1对数据A进行了修改并写回，事务2也对A进行了修改并写回，此时事务2写回的数据会覆盖事务1写回的数据，就丢失了事务1对A的更新。即对数据A的更新会被覆盖。

不可重复读：事务2读A，而后事务1对数据A进行了修改并写回，此时若事务2再读A，发现数据不对。即一个事务重复读A两次，会发现数据A有误。

读脏数据：事务1对数据A进行了修改后，事务2读数据A，而后事务1回滚，数据A恢复了原来的值，那么事务2对数据A做的事是无效的，读到了脏数据。

◆ 丢失更新

T1	T2
①读A=10 ② ③A=A-5写回 ④	读A=10 A=A-8 写回

◆ 不可重复读

T1	T2
①读A=20 读B=30 求和=50 ② ③读A=70 读B=30 求和=100 (验算不对)	读A=20 A←A+50 写A=70

◆ 读“脏”数据

T1	T2
①读A=20 A←A+50 写回70 ② ③ROLLBACK A恢复为20	读A=70

X锁是排它锁(写锁)。若事务T对数据对象A加上x锁，则只允许T读取和修改A，其他事务都不能再对A加任何类型的锁，直到T释放A上的锁。

S锁是共享锁(读锁)。若事务T对数据对象A加上S锁，则只允许T读取A，但不能修改A，其他事务只能再对A加S锁(也即能读不能修改)，直到T释放A上的S锁。

共分为三级封锁协议，如下：

- 一级封锁协议：事务在修改数据R之前必须先对其加X锁，直到事务结束才释放。可解决丢失更新问题。

T1	T2
① 对A加写锁 ② ③ 读A=10 ④ A=A-5写回 ⑤ 释放对A的写锁 ⑥ ⑦ ⑧	对A加写锁 等待 等待 等待 读A= 5 A=A-8 写回 释放对A的写锁



- 二级封锁协议：一级封锁协议的基础上加上事务T在读数据R之前必须先对其加S锁，读完后即可释放S锁。可解决丢失更新、读脏数据问题。

T1	T2
① 对A加写锁 ② 读A=20 ③ $A \leftarrow A+50$ ④ 写回70 ⑤ ⑥ ROLLBACK ⑦ A恢复为20 ⑧	对A加读锁 等待 等待 读A=20 释放对A的读锁

- 三级封锁协议：一级封锁协议加上事务T在读取数据R之前先对其加S锁，直到事务结束才释放。可解决丢失更新、读脏数据、数据重复读问题。

T1	T2
① 对A与B加S锁（读锁） 读A=20 读B=30 求和=50 ② ③读A=20 读B=30 求和=50 释放对A和B的读锁	对A加X锁（写锁） 注：由于A已加了读锁，所以等待 等待 等待 等待 读A=20 A←A+50 写A=70 释放对A的写锁



2025年下：综合知识



49、在封锁协议中，事务获取共享锁（S锁）后，对数据的操作权限是（ ）

- A.可读不可写
- B.可读可写
- C.不可读不可写
- D.不可读可写



分布式数据库（Distributed Database）是指数据存储和处理在多台计算机上分布式管理的数据库系统。它通过网络将多个数据库节点连接在一起，形成一个协同工作的整体，旨在提供数据的高可用性、可扩展性以及可靠性。

分布式数据库的特点：

- **数据分布**：数据不是集中存储在单个服务器上，而是分布在多个物理节点上。每个节点可以存储数据的一个子集，或者可能存储数据库的完整副本（如在高可用性系统中）。
- **透明性**：分布式数据库通常对用户和应用程序透明，用户在访问数据库时，不需要知道数据是存储在哪个节点上。这种透明性包括：
 - 位置透明性：用户不关心数据存储的具体位置。
 - 访问透明性：用户可以通过统一的接口访问所有数据。
 - 故障透明性：系统能够自动处理部分节点故障，确保系统的连续性。
- **数据冗余和复制**：为了增强系统的容错能力，分布式数据库可以通过数据复制的方式，在多个节点上保存相同的数据副本，确保数据的高可用性。如果某个节点故障，其他节点的副本可以继续提供服务。
- **数据一致性**：在分布式环境中，如何保证数据的一致性是一个挑战。常见的解决方案包括使用一致性协议（如两阶段提交协议（2PC）或Paxos协议），以及通过CAP定理来做出权衡：一致性、可用性和分区容忍性。
- **可扩展性、高可用性和容错性**：通过冗余和故障转移机制，分布式数据库能够确保系统在部分节点或网络故障的情况下仍能正常工作。



数据库系统-分布式数据库的优缺点



优点:

- 高可用性: 通过冗余和故障恢复机制, 分布式数据库可以提供比传统集中式数据库更高的可用性。
- 可扩展性: 随着数据量增长, 能够通过增加节点进行水平扩展, 轻松应对大规模数据存储和处理的需求。
- 容错性: 系统能够容忍单点故障, 保证数据的可靠性和系统的持续可用性。

缺点:

- 一致性问题: 分布式系统需要处理不同节点之间的数据一致性问题, 特别是在存在网络延迟或节点故障时。解决方案如分布式事务或一致性协议增加了系统的复杂性。
- 性能瓶颈: 虽然可以水平扩展, 但由于网络延迟、节点间的数据同步等因素, 分布式数据库的性能可能不如单节点的集中式数据库。
- 管理复杂性: 分布式数据库的管理需要处理多个节点的协调、故障恢复、数据备份、负载均衡等问题, 管理起来较为复杂。



分布式事务是指事务的参与者、支持事务的服务器、资源服务器以及事务管理器分别位于不同的分布式系统的不同节点之上。例如在大型电商系统中，下单接口通常会扣减库存、减去优惠、生成订单 id, 而订单服务与库存、优惠、订单 id 都是不同的服务，下单接口的成功与否，不仅取决于本地的 db 操作，而且依赖第三方系统的结果，这时候分布式事务就保证这些操作要么全部成功，要么全部失败。本质上来说，**分布式事务就是为了保证不同数据库的数据一致性。**

- 强一致性：这种一致性级别是最符合用户直觉的，它要求系统写入什么，读出来的也会是什么，用户体验好，但实现起来往往对系统的性能影响大，比较难实现。
- 弱一致性：这种一致性级别约束了系统在写入成功后，不承诺立即可以读到写入的值，也不承诺多久之后数据能够达到一致，但会尽可能地保证到某个时间级别（比如秒级别）后，数据能够达到一致状态。
 - 最终一致性：是分布式系统中一个重要的概念，指的是在系统中各个节点（或副本）在经过一段时间后，数据会趋向一致性，即所有的副本最终会达到一致的状态。最终一致性是一种放宽一致性要求的策略，常用于高可用性和扩展性需求较高的系统中。
- 注意：弱一致性是一个宽泛的大类，它的核心只有一个约束：不满足强一致性，而最终一致性是弱一致性这个大类里的一个具体、有明确承诺的子类：它不仅属于非强一致，还额外承诺了无论延迟多久，最终所有节点的数据一定会趋向一致。

那么，我们该如何保证不同数据库的数据一致性呢？



➤ 数据同步复制

- 原理：主节点执行数据更新后，必须等待所有副本节点完成数据写入并返回确认，才向客户端反馈操作成功。例如银行转账时，需确保所有记账节点都记录成功后，才告知用户转账完成。
- 优点：强一致性，所有副本与主节点数据实时一致，适合金融交易、财务系统等对数据准确性要求极高的场景。
- 缺点：性能受副本数量和网络延迟影响显著。若副本过多或网络卡顿，客户端请求可能长时间阻塞，降低系统吞吐量。

➤ 数据异步复制

- 原理：主节点更新后立即向客户端返回成功，后台通过异步线程将数据推送给副本。例如社交平台发布动态，发布按钮点击后立即显示成功，实际内容同步给其他服务器的过程在后台进行。
- 优点：响应速度快，主节点无需等待副本，适合微博、朋友圈等对实时性要求高但可容忍短暂数据不一致的场景（如用户刷新页面时可能看到旧数据，但几秒后会同步）。
- 缺点：主节点故障时可能丢失未同步的数据（如主节点更新后突然宕机，未同步到副本的交易记录会丢失），且副本间可能出现短暂数据不一致。



数据库系统-分布式数据库的事务



➤ 事件驱动架构

- 使用发布-订阅模型，数据变更时触发事件（如用户修改密码），事件携带变更信息并推送给订阅该事件的副本节点。副本节点接收到事件后，执行对应的更新操作。
- 注意事项：需保证事件不丢失（如使用消息队列持久化事件），否则副本可能遗漏更新（如事件中间件故障导致事件丢失，副本数据永久不一致）。

➤ 冲突解决（基于时间戳）

- 为每个数据写入操作附加时间戳，当多节点同时修改同一数据时，保留时间戳最新的版本。
- 适用场景：适用于笔记编辑、文档协作等允许“后改覆盖先改”的场景（如多人同时编辑在线文档，最终以最后保存的版本为准）。

➤ 数据一致性检查

- 定期扫描所有副本数据，通过哈希值比对或全量字段对比，发现不一致时以主节点或多数副本数据为准进行修复。例如每天凌晨对用户账户余额进行全量校验，若发现某副本余额与主节点不一致，自动同步主节点数据。



➤ 分布式一致性协议

- 2PC（两段提交协议）：用于分布式事务中的一致性协议，通过协调者节点向各参与者发送提交指令，确保所有节点要么提交事务，要么回滚。但2PC存在阻塞和单点故障的问题，参与者在协调者故障时可能会长时间等待。
- 3PC（三段提交协议）：是2PC的增强版，增加了“预提交”阶段，旨在解决2PC中的阻塞问题。3PC通过分阶段确认参与者的状态，提高了容错性，避免了协调者故障时的长时间阻塞，但实现复杂性相对较高。
- Paxos协议：经典的分布式一致性协议，但实现复杂，难以理解。
- Raft协议：相比于Paxos协议，Raft协议的设计目标是简化管理和实现，专注于为分布式系统提供强一致性保障。Raft通过**领导选举和日志复制**的机制来保持数据一致性，确保系统在节点故障或网络分区时依然能够正常运行。Raft的核心思想是将复杂的分布式一致性问题分解为易于理解的子问题，易于实现且能够提供较高的容错性，广泛应用于现代分布式系统中，如Kubernetes和Consul。

01

2PC一致性协议

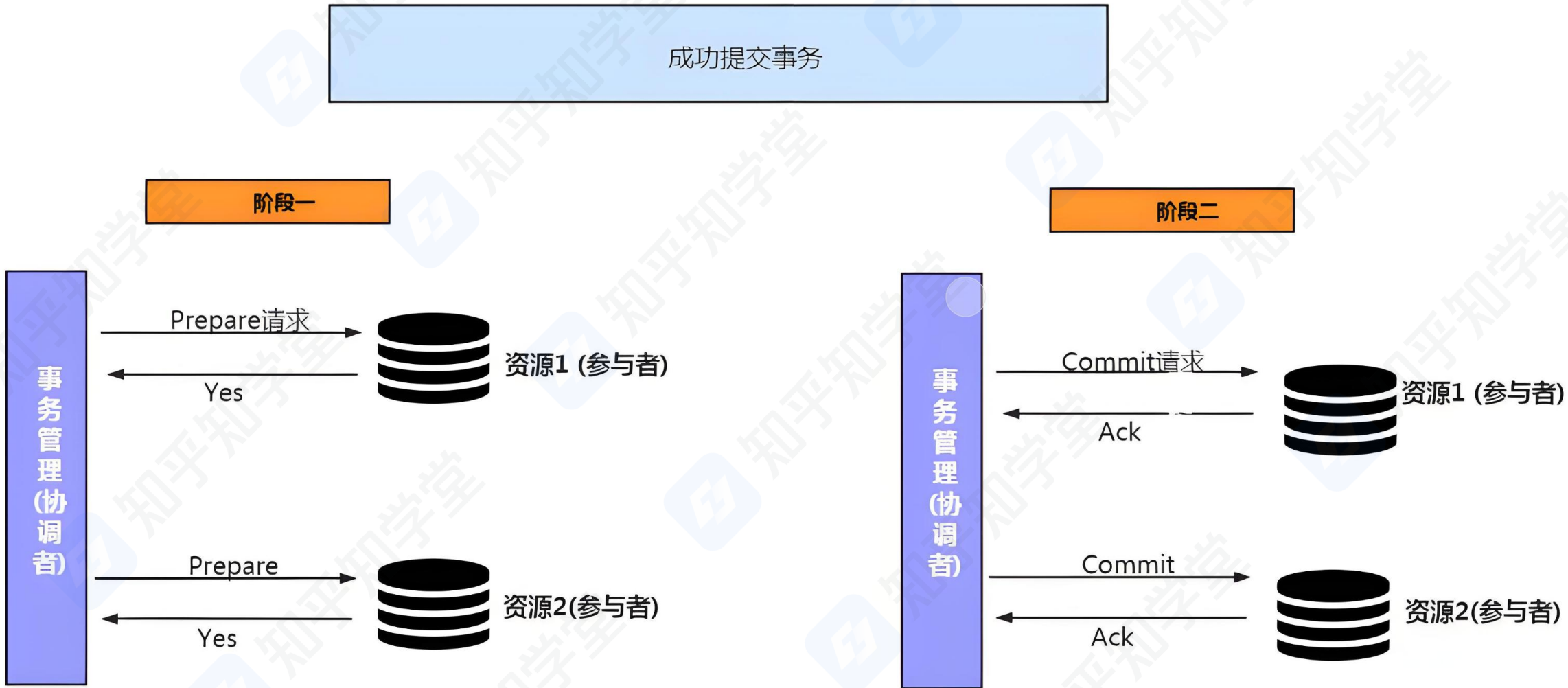
2PC是一种用于分布式事务的一致性协议，旨在确保在分布式系统中进行事务操作时，所有节点要么全都提交，要么全都回滚，保持系统的一致性。

它将整个事务流程分为两个阶段，准备阶段（**Prepare phase**）、提交/回滚阶段（**Commit/Rollback Phase**）

02

准备阶段：协调者向所有参与者节点（即系统中的各个数据库实例或存储节点）发送准备（prepare）请求，要求它们准备提交事务。每个参与者收到请求后，会检查事务是否能成功执行（比如没有约束冲突、数据一致性问题）。如果没有问题，参与者会锁定相关资源，准备提交并回复"同意"（Yes）；如果有问题，参与者会返回"拒绝"（No）。

提交/回滚阶段：如果所有参与者都同意提交（即所有参与者都返回"Yes"），协调者向所有参与者发送提交（commit）请求，要求它们正式提交事务并释放锁定的资源。如果任何参与者拒绝提交（即任何参与者返回"No"），协调者会向所有参与者发送回滚（rollback）请求，要求它们放弃事务并恢复到事务开始前的状态。



优点:

简单易理解: 2PC协议的设计简单, 只有两个阶段 (准备阶段和提交/回滚阶段), 这使得它相较于其他一致性协议 (如Paxos) 更容易理解和实现, 并且不涉及复杂的选举、日志复制等机制。这使得它可以较容易地被集成到现有的分布式数据库和事务系统中, 特别是在小规模分布式事务中。

保证原子性: 2PC能够保证事务的原子性, 即要么整个事务成功提交, 要么完全回滚。通过协调者控制提交流程, 确保所有参与者要么一致地提交事务, 要么一致地回滚, 避免了事务部分完成导致的数据不一致问题。

缺点:

同步阻塞: 二阶段提交协议存在最明显也是最大的一个问题就是同步阻塞, 在二阶段提交的执行过程中, 所有参与该事务操作的逻辑都处于阻塞状态, 也就是说, 各个参与者在等待其他参与者响应的过程中, 无法进行其他操作。这种同步阻塞极大的限制了分布式系统的性能。

单点故障: 协调者在整个二阶段提交过程中很重要, 如果协调者在提交阶段出现问题, 那么整个流程将无法运转, 更重要的是: 其他参与者将会处于一直锁定事务资源的状态中, 而无法继续完成事务操作。



CAP 原则：又称之为**CAP 定理**，指的是在一个分布式系统中， **Consistency**（一致性）、 **Availability**（可用性）、 **Partition tolerance**（分区容错性），三者不可得兼。

- 一致性（**C**）：在分布式系统中的所有数据备份，在同一时刻是否同样的值。（等同于所有节点访问同一份最新的数据副本）
- 可用性（**A**）：在集群中一部分节点故障后，集群整体是否还能响应客户端的读写请求。（对数据更新具备高可用性）
- 分区容错性（**P**）：以实际效果而言，分区相当于对通信的时限要求。系统如果不能在时限内达成数据一致性，就意味着发生了分区的情况，必须就当前操作在 **C** 和 **A** 之间做出选择。

CAP 原则的精髓就是要么 **AP**，要么 **CP**，要么 **AC**，但是不存在 **CAP**。如果在某个分布式系统中数据无副本，那么系统必然满足强一致性条件，因为只有独一数据，不会出现数据不一致的情况，此时 **C** 和 **P** 两要素具备，但是如果系统发生了网络分区状况或者宕机，必然导致某些数据不可以访问，此时可用性条件就不能被满足，即在此情况下获得了 **CP** 系统，但是 **CAP** 不可同时满足。

注：后面还有个**BASE**理论，它是对**CAP**理论的延伸，这里不再展开讲了



NoSQL: Non-Relational或者是Not Only SQL, 泛指非关系型数据库, 区分开关系型数据库, 并且不保证关系型数据库的ACID特性。

NoSQL的分类:

- 列式存储数据库: 跟传统的关系型数据库一样, 数据按行列进行存储, 这种类别通常用来应对分布式数据库的存储海量数据, 比如HBase
- 键值对存储数据库: 以key-value的形式来存储数据, 特点是简单、容易部署, 比如Redis
- 文档型数据库: 类似于键值对数据库, 可以看成是键值对数据库的升级版, 允许嵌套键值, 处理复杂数据的时候比传统的键值对存储效率高, 比如MongoDB
- 图数据库: 使用灵活的图形模型来存储数据, 能够拓展到多个服务器上, 适合存储通过图进行建模的数据, 比如社交网络、交通网络等, 常见的产品有Neo4J

NoSQL的特征: 易拓展、大数据量、高性能、灵活的数据模型、高可用

NoSQL的框架分层(从下至上): 数据持久层、数据分布层、数据逻辑模型层和接口层, 层次之间相辅而成, 协调工作

NoSQL适用于哪些场景: 数据模型比较简单、需要灵活性更强的系统、对数据性能要求高、不需要高度的数据一致性



数据库系统-NoSQL分类



键值对数据库

- Redis、MemCache
- 应用于内容缓存、处理大数据量的高访问负载、日志等
- 查找速度快但是数据无结构化

文档型数据库

- ConthDB、MongoDB（基于分布式文件存储的数据库，C++编写，主要用于处理大量文档；它是一种介于关系型数据库和非关系型数据库的中间产品，是nosql中功能最丰富、最像关系型数据库的非关系型数据库）
- 应用于web应用
- 数据结构要求不严格、表结构可变、不需要预定义表结构但查询性能不高且缺少统一查询语言

列存储数据库

- HBase（大数据）、Doris
- 应用于分布式文件系统
- 查找速度快、可扩展性强但功能相对局限

图关系数据库（不是存图形，而是存关系，比如：朋友圈、社交网络、广告推荐）

- Neo4j、InfoGrid
- 应用于社交网络、推荐系统
- 可以利用图结构相关的算法但是计算时需要全部图，导致不太好做分布式集群



T H E E N D

功不唐捐，玉汝于成！

• 开 启 新 征 程 •

