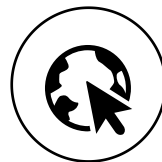


NEW PLAN

软考高级架构师

• — 段 新 征 程 •





大纲介绍





历年真题考点分析

时间	所属范围	考察知识点
2023 年 11 月 试题 1	软件架构	大数据架构 Lambda 和 Kappa
2023 年 11 月 试题 2	系统开发	SysML 需求图和用例图、需求图七类关系等
2023 年 11 月 试题 3	数据库	读写分离架构、redis 缓存、主从复制
2023 年 11 月 试题 4	Web 应用	Hibernate 架构、数据持久层、jwt
2023 年 11 月 试题 5	嵌入式	数字孪生概念、技术选择、架构图填空
2022 年 11 月 试题 1	软件架构	架构风格，质量属性
2022 年 11 月 试题 2	系统开发	结构化分析：数据流图、ER 图、数据字典
2022 年 11 月 试题 3	嵌入式	宇航装备架构、看图填空、故障分析
2022 年 11 月 试题 4	数据库	同步和异步、缓存分片、布隆过滤器
2022 年 11 月 试题 5	Web 应用	MQTT 协议、看图填空、云计算、边缘计算
2021 年 11 月 试题 1	软件架构	架构风格，质量属性
2021 年 11 月 试题 2	系统开发	用例图、顺序图填空、模型对比
2021 年 11 月 试题 3	软件架构	数据定义分布管理涵义、基于 FACE 的架构（题目不全）
2021 年 11 月 试题 4	数据库	反规范化设计方法、数据不一致、redis 同步
2021 年 11 月 试题 5	Web 应用	云平台智能家居，看图填空，TCP/UDP 区别
2020 年 11 月 试题 1	软件架构	架构风格，质量属性
2020 年 11 月 试题 2	数据库	逻辑设计、关系模式、主键、超类实体、派生属性
2020 年 11 月 试题 3	嵌入式	需求到架构映射、FACE 架构
2020 年 11 月 试题 4	数据库	内存数据库 redis，内存淘汰机制
2020 年 11 月 试题 5	Web 应用	非功能性需求，SSM 框架，数据访问机制
2019 年 11 月 试题 1	软件架构	架构风格，质量属性
2019 年 11 月 试题 2	系统开发	数据流图求实体、加工、补充数据流；系统流程图区别
2019 年 11 月 试题 3	嵌入式	信息物理系统三层结构概念、填空；三类安全威胁
2019 年 11 月 试题 4	数据库	数据库读写并发操作、key/value 方案探讨
2019 年 11 月 试题 5	Web 应用	非功能性需求、分布式架构图、SQL 注入攻击



数据库系统-下午案例分析考点

数据库设计：E-R模型、关系模型

事务：并发的问题和封锁协议

数据库规范化：范式、键与约束

数据库其他知识：反规范化、数据库安全、数据库备份、分布式数据库、缓存数据库、数据库集群、NoSQL

2023年涉及到的考点：

- 上午：分值大概4-5分，考察了范式、SQL语句、三级模式两级映像
- 下午：读写分离、主从复制、Redis数据库



数据库系统-范式

范式

- **第一范式：**要求数据库表中的所有字段都是不可分割的原子值。通俗地说，第一范式就是表中不允许有小表的存在。
- **第二范式：**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某一个候选键。通俗地说，就是表中不能存在联合主键
- **第三范式：**在2NF的基础上，要求数据库表中的每个非主属性不依赖于其它非主属性。也就是说，数据表中的每一列都和主键直接相关，而不依赖于其它列，即不能存在传递依赖
- **BC范式：**也称之为第三范式的补充范式，在3NF的基础上，要求数据库表中的所有属性都只依赖主键。也就是说，3nf要求每一个非主属性既不传递也不部分依赖于任一个码，而bcnf在此基础上要求主属性也不能传递或部分依赖于码，即：bcnf中不允许有除了码之外的其它元素之间存在任何函数关系
- **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。这个表不满足4NF，因为在这个表中，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。



数据库系统-主从数据库和读写分离

主从数据库架构是一种数据库冗余解决方案，它允许一个数据库服务器（称为“主”服务器）复制数据到一个或多个数据库服务器（称为“从”服务器）。在这种架构中，所有的数据更新操作（如INSERT、UPDATE、DELETE）首先在主服务器上执行，然后这些更改会被同步到从服务器。

特点包括：

- **数据冗余和读取负载均衡**：通过部署多个从服务器，可以在从服务器上处理读取请求，从而减轻主服务器的负载。
- **高可用性**：如果主服务器失败，可以快速将从服务器提升为新的主服务器，以确保服务继续运行。
- **备份和恢复**：从服务器可以用于备份，避免在主服务器上执行备份操作影响性能。

读写分离是数据层面的一种优化工程，目的是将数据库的读操作和写操作分散到不同的服务器上。通常结合主从复制技术一起使用，其中主数据库处理写请求，而从数据库则处理读请求。

主要优势包括：

- **提高并发性能**：分开处理读和写操作可以减少锁竞争，提高数据库的并发处理能力。
- **扩展性**：能够通过添加更多的从数据库来水平扩展读取能力，适应大量的读请求。
- **降低延迟**：读请求可以由最接近用户的服务器处理，减少了数据访问的延迟时间。

在实际应用中，主从数据库架构常常与读写分离技术结合使用。主数据库负责处理关键的、不可避免的写操作，而从数据库则处理大量的读取请求。这种方式可以有效地将数据的读写压力分散到不同的服务器，减少单点压力，并提高整个系统的性能与稳定性。

但是，这种架构也有其挑战，比如数据同步延迟可能导致从服务器数据与主服务器不一致，以及主服务器宕机时的故障转移处理等问题，这些都需要在设计时仔细考虑。此外，在实现读写分离时，还需要智能的数据库中间件或应用程序逻辑来确保读写请求正确地发送到合适的服务器。



数据库系统-主从数据库和读写分离

同步复制（Synchronous Replication）:在同步复制中，任何对主数据库的写操作都必须在从数据库上完成复制后才能确认完成。这确保了主数据库和从数据库始终保持一致。然而，这种方法可能会增加延迟，因为它要求在返回写操作成功之前，必须等待从数据库确认接收到数据。

半同步复制（Semi-Synchronous Replication）:半同步复制是同步复制和异步复制之间的折衷方案，其中写操作在主数据库上完成后会立即返回成功，但是主数据库在执行下一个写操作之前必须确保至少一个从数据库已经收到了前一个操作的数据。这可以在保持较低延迟的同时提供较强的数据一致性保证。

多版本并发控制（MVCC）:使用支持多版本并发控制的数据库系统可以减少读写冲突，每个读操作都可以访问到一致性时间点的数据快照，这样，读操作不会被写操作中的锁所阻塞，同时仍然能保证读取数据的一致性。

事务级别的一致性读取:使用事务来确保一致性是另一种方法。应用可以启动一个事务，在此事务中进行的读取将会看到写事务的最终结果，即使读取实际上是在从服务器上进行的。

数据库系统-主从数据库和读写分离

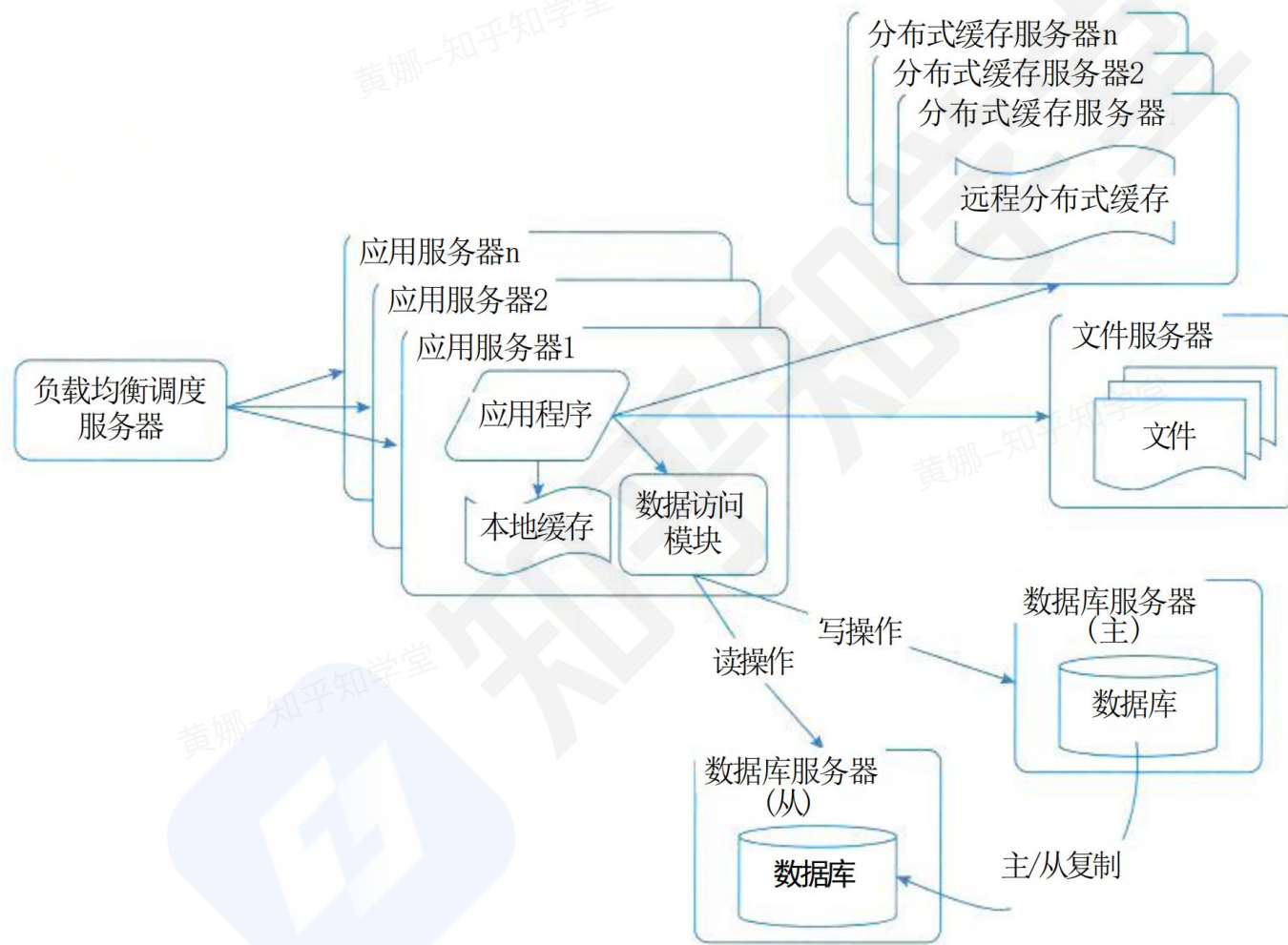


图10-14第五阶段网站架构



数据库系统-数据库分类



关系型数据库：关系数据库，是建立在关系模型基础上的数据库，借助集合代数等数学概念和方法来处理数据库中的数据。现实世界中的各种实体以及实体之间的各种联系均用关系模型来表示。简单说，关系型数据库是由多张能互相联接的二维行列表格组成的数据库。



NoSQL：泛指非关系型的数据库。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的特征：易拓展、大数据量、高性能、灵活的数据模型、高可用
- 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



NewSQL：用于描述一类结合了传统 SQL 数据库和 NoSQL 数据库的新型数据库系统。NewSQL 数据库旨在提供传统 SQL 数据库的严格数据一致性和复杂查询能力，同时具备 NoSQL 数据库的可伸缩性、高性能和分布式处理能力。

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

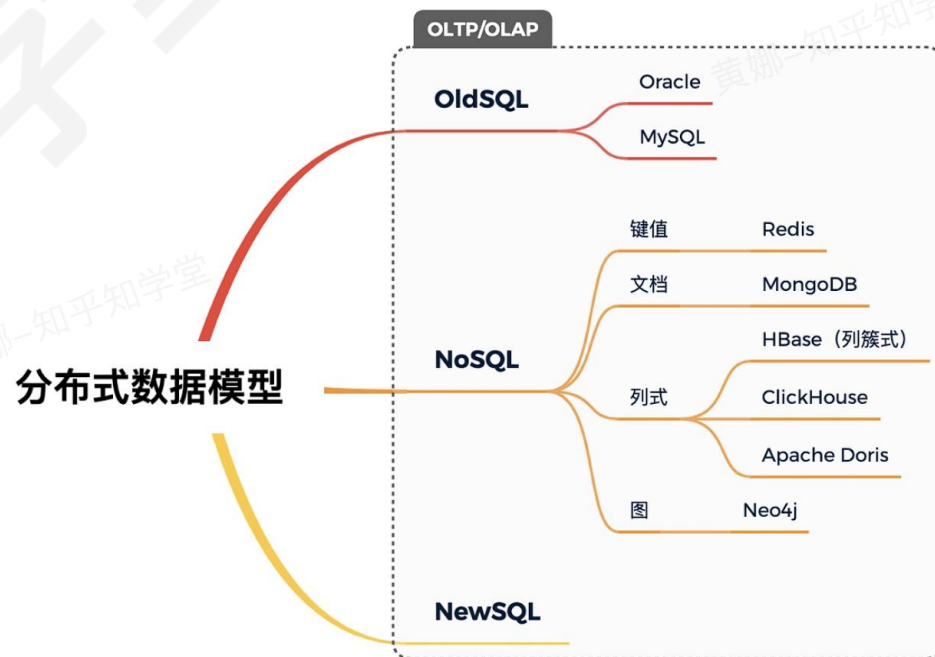
- ConthDB、MongoDB（基于分布式文件存储的数据库，C++编写，主要用于处理大量文档；它是一种介于关系型数据库和非关系型数据库的中间产品，是nosql中功能最丰富、最像关系型数据库的非关系型数据库）
- 应用于web应用
- 数据结构要求不严格、表结构可变、不需要预定义表结构但查询性能不高且缺少统一查询语言

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





数据库系统-缓存技术比较



内存数据库：将数据库整体存储在内存中，提高性能。

- **MemCache：** Memcache是一个高性能的分布式的内存对象缓存系统，用于动态Web应用以减轻数据库负载。Memcache通过在内存里维护一个统一的巨大的hash表，它能够用来存储各种格式的数据，包括图像、视频、文件以及数据库检索的结果等。
- **Redis：** Redis是一个开源的使用ANSI C语言编写、支持网络、可基于内存亦可持久化的日志型、Key-Value数据库，并提供多种语言的API。



Redis与Memcache的差异

- Redis和Memcache都是将数据存放在内存中，都是内存数据库。他们都支持key-value数据类型。同时Memcache还可用于缓存其他东西，例如图片、视频等等，Redis还支持list、set、hash等数据结构的存储。
- Redis中，并不是所有的数据都一直存储在内存中的。这是和Memcache相比一个最大的区别。当物理内存用完时，Redis可以将一些很久没用到的value交换到磁盘。
- Redis在很多方面支持数据库的特性，可以这样说他就是一个数据库系统，而Memcache只是简单地K/V缓存。



数据库系统-比较

特征	关系型数据库	NoSQL数据库	文件系统
数据模型	表格（表）结构，固定列和行	文档、列族、键值对、图形等	文件和目录结构
查询语言	SQL	数据库特定的查询语言或API	文件系统API
数据一致性	ACID事务支持	不同一致性模型（最终一致性等）	N/A
扩展性	垂直扩展（复杂），水平扩展（分片）	水平扩展（简单），适用于大规模数据	N/A
适用场景	复杂查询、事务性应用	高吞吐量、低延迟、大数据存储	文件和目录管理、存储和检索
数据复杂性	结构化数据	半结构化和非结构化数据	结构化和非结构化文件、目录结构
部署和维护	复杂，需要管理和维护	简化，容易扩展和部署	简化，通常由操作系统处理



数据库系统-分布式数据库



分布式数据库是一种将数据分布在多个节点或计算机上的数据库系统。这些节点通过网络连接，每个节点存储一部分数据，并可以独立运行，具有如下特点：

- 可扩展性：分布式数据库可以通过添加更多的服务器（节点）来扩展系统的存储和计算能力。这种水平扩展性使得分布式数据库能够应对大数据量和高并发请求的挑战。
- 高可用性和可靠性：由于数据分散存储在多个服务器上，即使其中一台或几台服务器出现故障，其他服务器仍可正常工作。通过备份和冗余策略，分布式数据库可提供更高的数据可用性和可靠性。
- 分布式处理：分布式数据库可以并行处理多个查询和事务，提升系统性能，缩短响应时间。
- 数据局部性：通过将数据存储在地域接近最常访问它的用户的服务器，分布式数据库可以减少网络延迟，提升性能。
- 灵活性：分布式数据库允许在不同的服务器、操作系统和网络环境中存储和处理数据。



数据库系统-数据仓库



数据仓库集成是把多种来源的数据集中在一起，建立数据仓库，所有数据都驻留在单个数据库服务器上，配置大型处理器和存储容量。数据仓库主要用于决策支持，在数据处理过程中强调分析。其特点是：

- 集成的数据：数据仓库中的数据来自多个来源，包括交易系统、运营系统、外部数据源等。这些数据经过清洗、整合和转换，使其具有统一的格式、结构和语义。数据集成是数据仓库建设的基础，它可以消除数据冗余和不一致，提高数据质量。
- 面向主题：数据按照主题组织，例如销售、客户、产品等。每个主题的数据都存储在一个单独的维度表和事实表中。面向主题的组织方式使数据仓库更加易于理解和使用。
- 数据相对稳定：数据一旦被加载，就不会经常更新。数据仓库主要用于分析历史数据，因此数据相对稳定是数据仓库的重要特征
- 包含历史信息：数据仓库存储了大量历史数据，这些数据可以用于分析趋势、发现模式和做出决策。历史数据是数据仓库的核心价值所在。



数据仓库的结构通常包含四个层次：

- 数据源：是数据仓库系统的基础，是整个系统的数据源泉。
- 数据的存储与管理：是整个数据仓库系统的核心。
- OLAP(联机分析处理)服务器：对分析需要的数据进行有效集成，按多维模型组织，以便进行多角度、多层次的分析，并发现趋势。
- 前端工具：主要包括各种报表工具、查询工具、数据分析工具、数据挖掘工具以及各种基于数据仓库或数据集市的应用开发工具。



商业智能：

- BI系统主要包括数据预处理、建立数据仓库、数据分析和数据展现四个主要阶段。



数据库系统-并发控制



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



数据库系统-封锁协议



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，其它任何事务都不能再对A加任何类型的锁，直到T释放A上的锁。这就保证了其它事务在事务T释放A上的锁之前不能再读取和修改A。



S锁是共享锁(读锁)。若事务T对数据对象A加上S锁，则其它事务只能再对A加S锁，而不能加X锁，直到T释放A上的S锁。这就保证了其它事务可以读A，但在事务T释放A上的S锁之前不能对A做任何修改。



共分为三级封锁协议，如下：

- 一级封锁协议**：事务T在修改数据A之前必须先对其加X锁，直到事务结束才释放。事务结束包括正常结束（COMMIT）和非正常结束（ROLLBACK）。**1级封锁协议可防止丢失修改，并保证事务T是可恢复的**。在1级封锁协议中，如果仅仅是读数据不对其进行修改，是不需要加锁的，所以它不能保证可重复读和不读"脏"数据

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



数据库系统-封锁协议

- 二级封锁协议：一级封锁协议的基础上加上事务T2在**读数据A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



数据库系统-封锁协议

- 三级封锁协议：一级封锁协议加上事务T在读取数据A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 $A \leftarrow A + 50$ 写A=70 释放对A的写锁



数据库系统-反规范化技术



反规范化技术：规范化设计后，数据库设计者希望牺牲部分规范化来提高性能。



采用反规范化技术的益处：降低连接操作的需求、降低外码和索引的数目，还可能减少表的数目，能够提高查询效率。



具体方法：

- 增加冗余列：在多个表中保留相同的列，通过增加数据冗余减少或避免查询时连接操作。
- 增加派生列：在表中增加可以由本表或其它表中数据计算生成的列，减少查询的连接操作并避免计算或使用集合函数。
- 重新组表：如果许多用户需要查看两个表连接出来的结果数据，则把这两个表重新组成一个表来减少连接而提高性能。
- 水平分割表：根据一行或多列数据的值，把数据放到多个独立的表中，主要用于表数据规模很大、表中数据相对独立或数据需要存放到多个介质上时使用。
- 垂直分割表：对表进行分割，将主键与部分列放到一个表中，主键与其它列放到另一个表中，在查询时减少I/O次数。



数据库系统-反规范化技术



反规范化技术可能带来的问题：数据的重复存储，浪费了磁盘空间;可能出现数据的完整性问题，为了保障数据的一致性，增加了数据维护的复杂性，会降低修改速度。



反规范化带来的数据不一致性问题主要有以下两种：

- 更新异常: 当需要更新冗余数据时，如果更新操作不同步，可能会导致数据不一致。
- 插入异常: 当向冗余表中插入新数据时，如果插入操作不符合相关规则，也可能导致数据不一致。



解决数据不一致性的几种常规解决方案：

- **应用触发器**: 触发器是一种存储过程，它会在对数据库中的数据进行增、删、改操作时自动执行。可以使用触发器来确保冗余数据的一致性。
- **使用并发控制机制**: 并发控制机制可以确保多个用户同时对数据库进行操作时数据的安全性一致性。
- **采用应用程序逻辑**: 应用程序逻辑也可以用来确保冗余数据的一致性。



选择合适的方法：

- 在实际应用中，可以根据具体情况选择合适的方法来解决反规范化带来的数据不一致性问题。一般来说，**如果数据更新频繁，则可以使用触发器或并发控制机制；如果数据更新不频繁，则可以使用应用程序逻辑。**



数据库系统-ORM



ORM，即Object-Relationl Mapping，它在关系型数据库和对象之间作一个映射，这样，我们在具体的操作数据库的时候，就不需要再去和复杂的SQL语句打交道，只要像平时操作对象一样操作即可。



面向对象编程把所有实体看成对象(object)，关系型数据库则是采用实体之间的关系(relation)连接数据。很早就有人提出，关系也可以用对象表达，这样的话，就能使用面向对象编程，来操作关系型数据库。



ORM把数据库映射成对象。

- 数据库的表(table) --> 类(class)
- 记录(record，行数据)-->对象(object)
- 字段(field) -->对象的属性(attribute)



ORM优点：

- 使用ORM可以大大降低学习和开发成本。
- 程序员不用再写SQL来进行数据库操作。
- 减少程序的代码量。
- 降低由于SQL代码质量差而带来的影响。



ORM缺点

- 不太容易处理复杂查询语句。
- 性能教直接用SQL差。



典型历年真题详解-2021年

阅读以下关于数据库设计的叙述，在答题纸上回答问题1至问题3。

【说明】

某医药销售企业因业务发展，需要建立线上药品销售系统，为用户提供便捷的互联网药品销售服务、该系统除了常规药品展示、订单、用户交流与反馈功能外，还需要提供当前热销产品排名、评价分类管理等功能。通过对需求的分析，在数据管理上初步决定采用关系数据库(MySQL)和数据库缓存(Redis)的混合架构实现。经过规范化设计之后，该系统的部分数据库表结构如下所示。

供应商（供应商ID，供应商名称，联系方式，供应商地址）；

药品（药品ID，药品名称，药品型号，药品价格，供应商ID）；

药品库存（药品ID，当前库存数量）；

订单（订单号码，药品ID，供应商ID，药品数量，订单金额）；

【问题1】(9分)

在系统初步运行后，发现系统数据访问性能较差。经过分析，刘工认为原来数据库规范化设计后，关系表过于细分，造成了大量的多表关联查询，影响了性能。例如当用户查询商品信息时，需要同时显示该药品的信息、供应商的信息、当前库存等信息。为此，刘工认为可以采用反规范化设计来改造药品关系的结构，以提高查询性能。修改后的药品关系结构为：

药品（药品ID，药品名称，药品型号，药品价格，供应商ID，供应商名称，当前库存数量）；

请用200字以内的文字说明常见的反规范化设计方法，并说明用户查询商品信息应该采用哪种反规范化设计方法。

【问题2】(9分)

王工认为，反规范化设计可提高查询的性能，但必然会带来数据的不一致性问题。请用200字以内的文字说明在反规范化设计中，解决数据不一致性问题的三种常见方法，并说明该系统应该采用哪种方法。



典型历年真题详解

【问题3】(7分)

该系统采用了Redis来实现某些特定功能（如当前热销药品排名等），同时将药品关系数据放到内存以提高商品查询的性能，但必然会造成Redis和MySQL的数据实时同步问题。

(1) Redis的数据类型包括String、Hash、List、Set和ZSet等，请说明实现当前热销药品排名的功能应该选择使用哪种数据类型。

(2) 请用200字以内的文字解释说明解决Redis和MySQL数据实时同步问题的常见方案。



典型历年真题详解-2021年

参考答案:

【问题1】

常见反规范化技术如下:

- (1)增加冗余列: 在多个表中保留相同的列, 通过增加数据冗余减少或避免查询时的连接操作。
- (2)增加派生列: 在表中增加可以由本表或其它表中数据计算生成的列, 减少查询时的连接操作并避免计算或使用集合函数。
- (3)重新组表: 如果许多用户需要查看两个表连接出来的结果数据, 则把这两个表重新组成一个表来减少连接而提高性能。
- (4)水平分割表: 根据一行或多行数据的值, 把数据放到多个独立的表中, 主要用于表数据规模很大、表中数据相对独立或数据需要存放到多个介质上时使用。
- (5)垂直分割表: 对表进行分割, 将主键与部分列放到一个表中, 主键与其它列放到另一个表中, 在查询时减少I/O次数。

用户查询商品信息采用的是增加冗余列的方式。

【问题2】

批处理维护、应用逻辑和触发器。

- (1)批处理维护: 指对复制列或派生列的修改积累一定的时间后, 运行一批处理作业或存储过程对复制或派生列进行修改, 这只能在实时性要求不高的情况下使用。
- (2)应用逻辑: 要求必须在同一事务中对所有涉及的表进行增、删、改操作。用应用逻辑来实现数据的完整性风险较大, 因为同一逻辑必须在所有的应用中使用和维护, 容易遗漏, 特别是在需求变化时, 不易于维护。
- (3)触发器: 对数据的任何修改立即触发对复制列或派生列的相应修改。触发器是实时的, 而且相应的处理逻辑只在个地方出现, 易于维护。一般来说, 是解决这类问题比较好的办法。

该系统应该采用触发器。



典型历年真题详解-2021年

【问题3】

(1)ZSet

(2)

一、对强一致要求比较高的，应采用实时同步方案，即查询缓存查询不到再从DB查询，保存到缓存；更新缓存时，先更新数据库，再将缓存的设置过期（建议不要去更新缓存内容，直接设置缓存过期）。

二、对于并发程度较高的，可采用异步队列的方式同步，可采用kafka等消息中间件处理消息生产和消费。

三、使用阿里的同步工具canal,canal实现方式是模拟mysql slave和master的同步机制，监控DB binlog的日志更新来触发缓存的更新，此种方法可以解放程序员双手，减少工作量，但在使用时有些局限性。

四、采用UDP自定义函数的方式，面对mysql的API进行编程，利用触发器进行缓存同步。

T H E E N D

功不唐捐，玉汝于成！

• 开 启 新 征 程 •

