road map
数据库:
还剩
9 查询执行
10 查询优化
6 物理数据库 没有理解。注意前两者和查询结构是紧密关联的，看的时候可以和前面结合一下
5.19
看完上面没理解的部分，然后从头开始复习，着重关系代数和 sql 查询（应用），ER 图和逻辑数据库设计（最后一个大题），证明题，**物理数据库**和后面联系比较紧密，主要是几个按图索骥的大题以及知识点的考察。最后两章，关键是知识点
5.20

1 绪论

数据库模式 数据库不是仅用一种模式来刻画的 数据库的三层模式结构 内模式/存储模式：描述数据库的物理存储结构和存取方法，对应物理数据模型 概念模式：为全体数据库用户描述整个数据库的结构和约束，对应实现数据模式 外模式/视图
数据模型的三个要素 描述数据库的一系列**概念** 操纵数据库的一系列**操作** 数据库应当服从的**约束条件**
数据模型的分类（按用途分） 概念数据模型 实现数据模型->关系数据模型 物理数据模型
模式映射 三层模式结构中，不同层次模式间的映射完成**应用程序**与**数据库**之间的**数据转换**和**请求转换** 外模式-概念模式映射 概念模式-内模式映射
数据独立性 逻辑数据独立性：概念模式改变，只需修改外模式-概念模式映射，而外模式和基于外模式开发的应用程序都不需要改变 物理数据独立性：内模式改变，只需修改概念模式到内模式的映射，概念模式和外模式均不需要改变
故障恢复：确保系统重启后数据库可以恢复到最近的一致性状态

2 关系数据库

关系数据模型 三个要素 1. 关系数据结构（关系） 2. 关系操作（查询和更新） 3. 关系完整性约束
关系的度：属性的数目
超键：可以唯一标识一个元组的一组属性 候选键：最小的超键 主键：人为指定候选键中的一个作为主键 外键：F 是关系 R（参照关系）的一个属性子集，他对应了关系 S（被参照关系）的主键 K，则称 F 是外键
完整性约束的类型 1. 实体完整性约束（主键值唯一且不为空） 2. 参照完整性约束（外键要么为空，要么不为空且 F 的值在 S 中存在） 3. 用户定义完整性约束（用户自定义的）
关系模式 Schema，提前定义，不经常变化 实例，动态变化

2.2 关系代数

一个关系代数表达式可以看作是一个函数，输入输出都是关系
关系代数查询基本都可以由基本关系代数操作组成

2.3 关系演算

能读懂就可以了

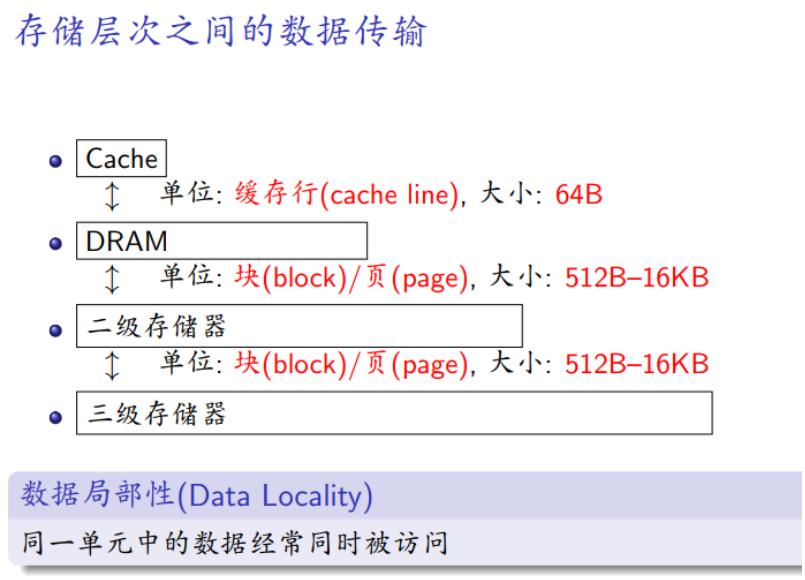
5 逻辑数据库设计

数据依赖：关系的属性在语义上的依赖关系 5.2-5.4 都是为了处理数据依赖的问题 5.2 是与数据依赖相关的概念 5.3 关系数据库规范化理论 用来评估关系模式的规范化程度（判断处于哪一个范式） 5.4 关系模式分解 消除数据依赖，判断关系模式分解是否满足无损连接性和函数依赖性
包含在任一候选键中的属性称为**主属性**，其他属性称为**非主属性**

6 物理数据库设计

7 存储管理

存储介质
基于磁盘的数据库存储结构（数据结构）
缓冲区管理
计算机系统的存储器被组织成**层次结构**
存储器的分类（根据 CPU 方问存储介质的方式分） ➢ 主存储器 ➢ 二级存储器 ➢ 三级存储器
主存储器 寄存器 高速缓存 内存 按**字节**寻址 CPU 可使用 load/store 直接访问
二级存储器 磁盘/机械硬盘 闪存/固态硬盘 按**块**寻址 **联机**使用 CPU 无法直接访问，只能先读入到主存储器中
三级存储器 磁带 光盘 网络存储 按**块**寻址 **脱机**使用



存储器的分类（按照易失性）
易失性存储器 主存储器
非易失性存储器 二级存储器 三级存储器

磁盘块/页：将格式化磁盘时，将他划分为许多大小相同的块（页），题目中通常会给出这个大小

数据库存储在磁盘中
在读/写之前，需要将数据库从磁盘读入内存
写数据后，将修改过的文件页写回磁盘，替换原有页

一个数据库存储为多个文件，一个文件包含多个页

数据库的页可以存储和不同的数据，如元组，元数据，索引，日志记录等等

DBMS 的**存储管理器**负责管理数据库文件
记录页中元组的读/写
记录页中的空闲空间

元组的表示
元组表示为**字节序列**
DBMS 根据元组所在的**关系的模式**（schema），将元组的字节序列翻译成元组的全部属性值
DBMS 在系统目录中记录关系模式

元组由两部分组成：元组头（记录元数据，如指向关系模式的指针，元组的长度等等）和元组数据（由元组的所有属性值拼接而成，每个属性值在字节序列中的偏移量是 4 字节或 8 字节的倍数）

变长元组：
定长属性值和变长属性值分别置于元组的两端
元组头后紧跟指针数组，指向每个属性值

页布局
一个页包括两部分：页头（元数据，如页的大小，页的校验和，槽的元数据（**槽的数量和最后一个槽的起始位置的偏移量**）等等）和页数据

页中数据的组织方法 面向元组的组织方法 日志结构的组织方法
分槽页 页头后面是槽数组，数组中的每个元素存储对应槽的起始位置的偏移量 每个元组占据一个槽，从后向前分布在页的末尾。起始位置的偏移量是 4 字节或 8 字节的倍数
DBMS 为一个关系中的每个元组分配唯一的**记录号**，有两种表示方法 （页号，槽号） 使用唯一的整数标识。DBMS 使用间接层，将元组 ID 映射为（槽号，页号）
文件组织 堆文件组织（堆文件中的元组以**任意顺序**存储） 顺序/有序文件组织（元组按**排序键的顺序**存储） 哈希文件组织（根据**元组键的哈希值**来确定元组存储在哈希文件的哪个页）
堆文件中页的组织方法 链表（数据页：存储元组的数据页连接成的链表。空闲页：所有空闲页组织成的链表。头页：存储以上两个链表的头指针以及其他的元数据信息） 页目录（数据页，页目录：记录每个数据页的位置和空闲空间信息。DBMS 需要保证页目录中的信息与实际数据页信息同步）
缓冲区管理器 负责在**磁盘和内存**之间复制文件页
DBMS 将**可用的内存区域**划分为**页数组**，称为**缓冲池**（也就是按照他自己的方式，把分给数据库的这块内存空间重新组织了一下） **页框**：缓冲池中的页
页表：记录缓冲池中当前有哪些页，以及这些页在内存中的地址（即将页号映射为一个地址）
页框的原数据：用两个变量记录每个页框的状态 pin_count：页框中的页当前**被请求但是未释放**的次数，即引用计数。（读/写） dirty：某个页被读入缓冲池后，是否被修改过
缓冲区管理器的功能： 请求页 如果在缓冲池中，则 pin_count += 1，并返回地址 如果不在缓冲池中，则找到缓冲池中一个 pin_count=0 的页，使用**页替换策略**，注意如果被替换的页是脏页，那么要写回。 如果缓冲池中没有 pin_count=0 的页，那么就要等待（实际上的实现是中止该事务，并重新执行他） 修改页 dirty=True 在被替换时，是实际修改磁盘中的页的时刻 释放页 pin_count -= 1

8 索引结构

首先掌握基本的概念，然后看以下三个算法的 数据结构 查找算法 插入算法

可扩展哈希表
线性哈希表
B+树

索引能够快速的找到关系中满足搜索条件的元组

索引实际上是一个根据某个属性（key）查找元组地址（value）的查找表

索引键：索引根据一组属性来定位元组
索引项：索引中的（键值，地址）对

索引分为有序索引（索引项根据索引键值来排序）和哈希索引（维护一个 key 到 value 的直接映射）

注意：哈希索引只能支持索引键上的**等值查找**，也就是不支持比大小的查找

有序索引的分类（根据数据文件（页）中的元组是否按照索引键排序）

聚簇索引	数据文件中的元组是按照索引键排序的 聚簇索引的**索引键**通常是关系的**主键** 通常只有一个聚簇索引-有两个的话，在数据文件中的排序就冲突了
非聚簇索引	数据文件中的元组不是按索引键排序的 一个关系上可以有很多个非聚簇索引

索引组织表=聚簇索引文件+数据文件
意思是将二者合二为一，在聚簇索引的索引项中，存储元组本身，而不是元组的地址。这样的方法在根据索引查找元组时，可以减少一次从磁盘的 IO

有序索引的分类（根据关系中的每个元组在索引中是否都有一个对应索引项）

稠密索引	非聚簇索引一定是稠密索引
稀疏索引	聚簇索引一般是稀疏索引 这是因为，在聚簇索引中，数据文件的元组按照索引键排序，因此相同索引键的元组必定相邻排列。因此可以在所以项中，只记录其首元组的地址，其他相同索引键值的可以通过扫描得到

有序索引的分类（根据索引键是否为关系的主键）

主索引	一个关系只有一个主索引（只有一个主键）
二级索引	通常是非聚簇索引 一个关系可以有多个二级索引

MySQL 中的索引
主索引是索引组织表
二级索引的索引项中存储的不是元组地址，而是主键值（在数据地址更新时，二级索引的索引项不需要同时改变）

唯一索引：索引键值不能重复
主索引（主键）一定是唯一索引
唯一索引一定是稠密索引

外键索引：索引键是关系的外键
加快参照完整性检查。被参照关系的元组删除时，根据其主键的值=外键的值，找到有没有外键的元组，如果有，则违背了参照完整性。被参照关系的元组修改时同理

索引结构
都是从四个角度来掌握：
数据结构
查找索引项的方法
插入索引项的算法
删除索引项的算法

Hash-based Index Structures	可扩展哈希表 线性哈希表
Tree-based Index Structures	B+树

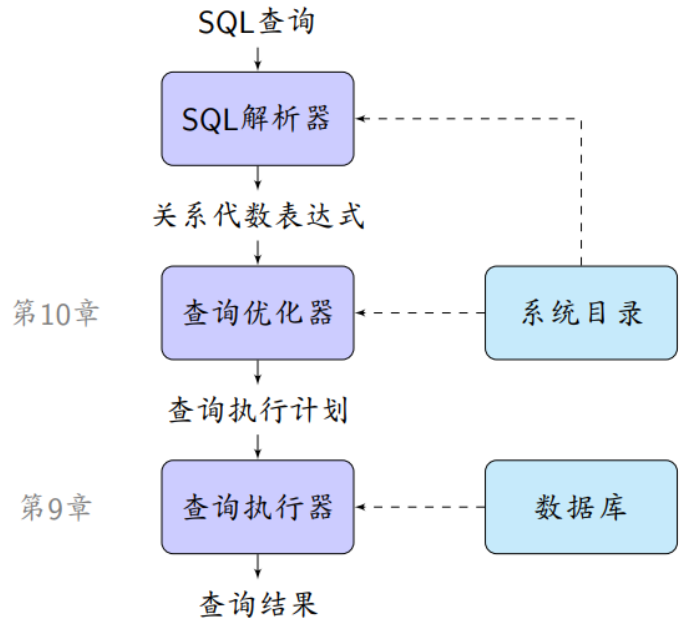
外存哈希表分为两类：静态哈希表（桶的数量固定不变）和动态哈希表（桶的数量动态变化，每个桶中的索引项存储在大约 1 个页中。可扩展哈希和线性哈希都是动态哈希）

B+树的性质
所有叶节点在同一层
除了根节点以外，每个节点至少半满
每个节点恰好放入一个页

每个叶节点包括一个索引项数组和指向右兄弟节点的指针
内节点包括一个键数组和一个指向儿子节点的指针数组

9 查询执行

查询处理的基本过程



SQL 解析器：SQL 查询->关系代数表达式
查询执行计划：用于执行一个查询的原语操作序列

外排序

按照**排序键**对元组进行排序
由于数据库存储在文件中，所以只能用**外排序** 外排序的时间开销主要在于对磁盘的访问，用**磁盘 I/O 数量**来近似衡量
两趟多路外存归并排序

选择操作的执行

选择算法	I/O 代价	可用内存要求	brief
基于扫描	B(R)	M>=1	
基于哈希	B(R)/V(R, K) 每个桶的平均块数	M>=1	只支持等值选择 R 采用哈希文件组织形式

			K 是 R 的哈希键
基于索引	聚簇索引：B（R）/V(R, K) 非聚簇索引：T(R)/V(R, K)	M>=1	等值或者区间选择（同 B+树）

确定连接操作的执行算法↵

- 一趟连接(One-Pass Join)：左关系可以全部读入缓冲池的可用页面↵
- 索引连接(Index Join)：左关系较小。右关系在连接属性上建有索引↵
- 排序归并连接(Sort-Merge Join)：至少有一个关系已经按连接属性排序。多个关系在相同连接属性上做多路连接也适合使用排序归并连接。↵
- 哈希连接(Hash Join)：在一趟连接、排序归并连接、索引连接都不适用的情况下，哈希连接总是好的选择↵
- 嵌套循环连接(Nested-Loop Join)当内存缓冲区的可用页面特别少时，可使用嵌套循环连接

投影操作的执行

	I/O	可用内存	brief
不去重	B(R)	M>=1	在数据访问模式上和基于扫描的选择算法相同

去重操作

	I/O	可用内存	brief
一趟去重	B(R)	扫描占用一个页，其余的页必须能放下 R 中互不相同的元组（最终的结果）	扫描，用一个哈希表存储见过的元组
基于排序的去重	3B(R)（归并段写入写回 2，归并阶段 1）	B（R）<=M^2 同归并算法	和多路归并算法类似，只是排序时，按照元组的所有属性排序（而不是排序键）。归并时，相同的元组只输出一个
基于哈希的去重	3B(R)（分桶时，每个块读入一次，然后把每个桶写入文	B(R)<=(M-1)^2 一共有 M-1 个桶，并且在每个桶上执	先把元组按照哈希值分组（所有属性作为键值），然后对

	件（也是 B(R)个块），一趟去重 O（1））	行一趟去重算法时，可用内存页数也要满足需求	每一个桶进行一趟去重

聚集操作

本质上和**去重操作**相同，都是找相同的元素

集合差操作

	I/O	可用内存	brief
一趟集合差	B(R)+B(S)	B(S)<=M-1	先用一个哈希结构存储 S 的元组，然后扫描 R，把不在里面的去掉
基于排序的集合差	3B(R)+3B(S)	B(R)+B(S)<=M^2	分别对两个关系构造归并段，然后放到内存中，做集合差
基于哈希的集合差	3B(R)+3B(S)（相当于分了两次桶，最后计算集合差的复杂度时 B(R)+B(S)）	B(S)<=(M-1)^2 一共有 M-1 个桶，并且在每个桶上执行一趟去重算法时，可用内存页数也要满足需求	分别将两个关系放到桶里面，然后逐桶计算集合差。最终的结果取每个桶结果的并集

集合的并运算和交运算都可以由差运算推导出来，因此本质上是相同的

连接

只考虑自然连接

	I/O	可用内存	brief
一趟连接	B(R)+B(S)	B(S)<=M-1 内存查找结构的大小，相当于把 S 全部读入了内存	分为 build(构造哈希结构)和 probe（寻找可以连接的部分）阶段
嵌套循环连接算法	T(R)*（T(S)+1）(外关系每个元组读一次，内关系每个元	M>=2	两重循环查找相同的属性值

	组读 T(外关系)次)		
基于块的嵌套循环	B(S)+B(R)B(S)/(M-1)	M>=2	外关系的每 M-1 块同时读入内存，内关系也按照块读取
排序归并连接	3B(R)+3B(S)	B(S)+B(R)<=M^2	分别创建归并段，并按照同名属性进行排序，然后在归并时进行连接
经典哈希连接			一趟连接算法的内存查找结构使用的是哈希表
Grace 哈希连接	3B(R)+3B(S)	B(S)<=(M-1)^2	分桶构造哈希表（按照同名属性值），然后分桶连接，最后取并集
索引连接	B(R)+T(R)T(S)/V(S, Y)(索引是非聚簇索引，此时对于 R 中的每个元组，平均有 T(S)/V(S, Y))个 S 中的元组与他对应，因此要分别读取这些元组，产生这么多次 IO） B(R)+T(R)*B(S)/V(S, Y)	M>=2	关系 S 上建有索引。则遍历 R 的每条元组，然后通过索引查找有无对应的 S 中的元组

算法分析

输出结果时，产生的 I/O 不计入算法的 I/O 代价
算法分析的角度有两个: I/O 代价：根据算法可以得出（时间） 可用内存需求:（空间）

10 查询优化

基于代价的查询优化 **计划枚举**

代价计算
查询计划枚举 1. 关系代数表达式的等价变化 2. 连接顺序优化：确定连接操作的最优执行顺序
关系代数表达式等价：执行结果相同

11 并发控制

只考基于锁的并发控制 隔离级别不考
等价调度：如果两个调度在任何数据库实例上的效果都相同，则称这两个调度为等价调度
可串行化调度：如果一个调度等价于一个可串行调度，则称之为可串行化调度
冲突 如果两个**操作**（注意操作是比事务更小的粒度，一个事务由很多操作组成，而我们关心其中的读写操作）满足以下三个条件，则两个操作冲突 ➢ 属于不同事务 ➢ 涉及相同的对象 ➢ 其中至少有一个操作是写操作
事务中与**调度**相关的操作只有**读和写**
冲突等价 两个调度。如果 ➢ 涉及相同事务的相同操作 ➢ 每一对冲突的操作在两个调度中的顺序都相同
冲突可串行化：一个调度**冲突等价**于一个串行调度 如果可以通过将事务 S 中不同事务中的非冲突操作交换顺序，可以将 S 转换为一个串行调度，则 S 是一个冲突可串行化调度
判断是不是冲突可串行化： 两个事务的调度 把他对应的串行调度写出来，看是否满足冲突等价 多个事务的调度 将调度 S 表示为优先图（每个顶点是一个事务，有向边表示事务 A 的某个操作 a1 和事务 B 的某个操作 b1 冲突，并且 a1 在 b1 的前面） 如果没有环，则是冲突可串行化，并且这个图的拓扑排序表示了与其等价的串行调度
并发控制协议：对并发事务实施正确的（运行时）调度，而无需预先确定整个（静态）调度

12 故障恢复

故障的类型

事务故障	逻辑错误：事务由于内部错误而无法完成，比如违反完整性约束 内部状态错误：DBMS 由于内部状态错误（死锁）必须中止活跃事务
系统故障	软件故障：DBMS 实现的 bug 所导致的故障 硬件故障：运行 DBMS 的计算机发生崩溃
存储介质故障	非易失存储器发生故障，损坏了存储的数据 假设数据损坏可以被检测，如使用校验和（checksum） 任何 DBMS 都无法从这种故障中恢复，只能从备份中还原

故障恢复的作用

数据库日志：DBMS 在数据文件之外维护的一个日志文件，用于记录事务对数据库的修改

DBMS 在进行故障恢复时执行的两种操作
Undo 撤销未完成的事务对数据库的修改
Redo 重做已提交事务对数据库的修改

DBMS 如何运用 undo 和 redo 取决于 DBMS 如何管理缓冲池，缓冲池策略由 STEAL 和 FORCE 组成

STEAL/NO-STEAL 策略
DBMS 是否允许将未提交事务所做的修改写到磁盘并覆盖现有数据
STEAL：允许
NO-STEAL：不允许

FORCE/NO-FORCE 策略
DBMS 是否强制事务在提交前必须将所做的修改全部写回磁盘

缓冲池效率高	STEAL + FORCE	STEAL + NO-FORCE
缓冲池效率低	NO-STEAL + FORCE	NO-STEAL + NO-FORCE
	I/O效率低	I/O效率高

NO-STEAL+FORCE 无需 undo 和 redo
优点：简单
缺点：缓冲池必须足够大，得以存储下所有的修改

缓冲池策略+日志文件

缓冲池策略不同，WAL 协议不同，那么日志文件的记录方法也不同