数据库系统概述

1.1 基本概念*（概念）*

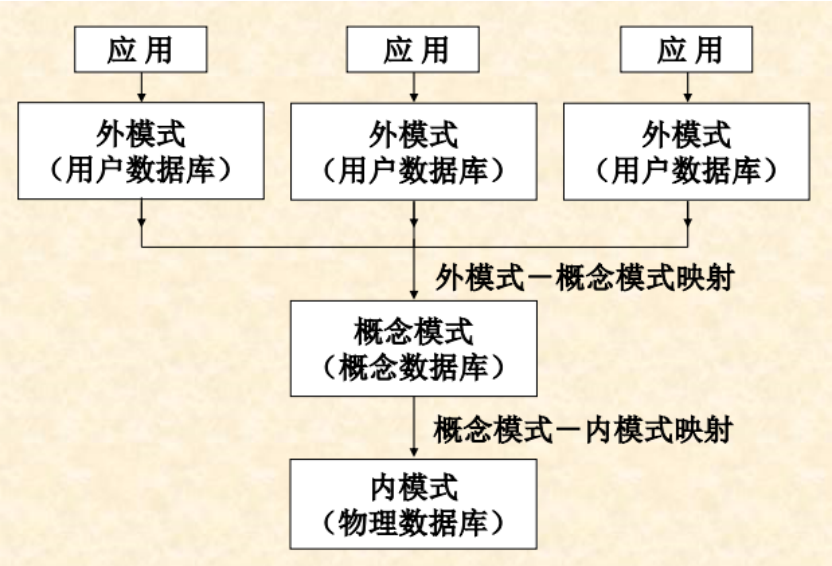
- 数据库，数据库管理系统，数据库系统，数据库管理员
  - 数据库DB：是**数据集合**，具有**统一的结构形式**并存放于**统一的存储介质**内，它由多种应用数据集成，并可被应用所共享；**结构化、集中存储、数据共享**
  - 数据库管理系统DBMS：**是一种管理数据库的系统软件**；数据组织、数据操纵、数据维护、数据控制和保护、数据交换、数据服务、数据字典；DDL数据定义语言、DML数据操纵语言、DCL数据控制语言
  - 数据库系统：数据库、数据库管理系统、数据库管理员、软件平台、硬件平台
  - 数据库管理员DBA：对数据库进行规划、设计、维护、监视的专职人员

1.3 数据库系统的特点（概念）

- **数据的集成性：集多种应用数据于一体**
- 数据的**高共享性**与**低冗余性**：共享--供多个应用程序使用；冗余--同一数据重复存储
- **数据独立性**（物理、逻辑）：物理--数据的物理结构的改变，不影响数据库的逻辑结构，从而不致引起应用程序的变化；逻辑--数据库总体逻辑结构的改变，不需要相应修改应用程序
- 数据的统一管理与控制（**完整性检查、安全性保护、并发控制、故障恢复**）
  - 完整性：数据正确性
  - 安全性：防止非法访问
  - 并发控制
  - 故障恢复

1.4 数据库内部体系结构*（概念）*

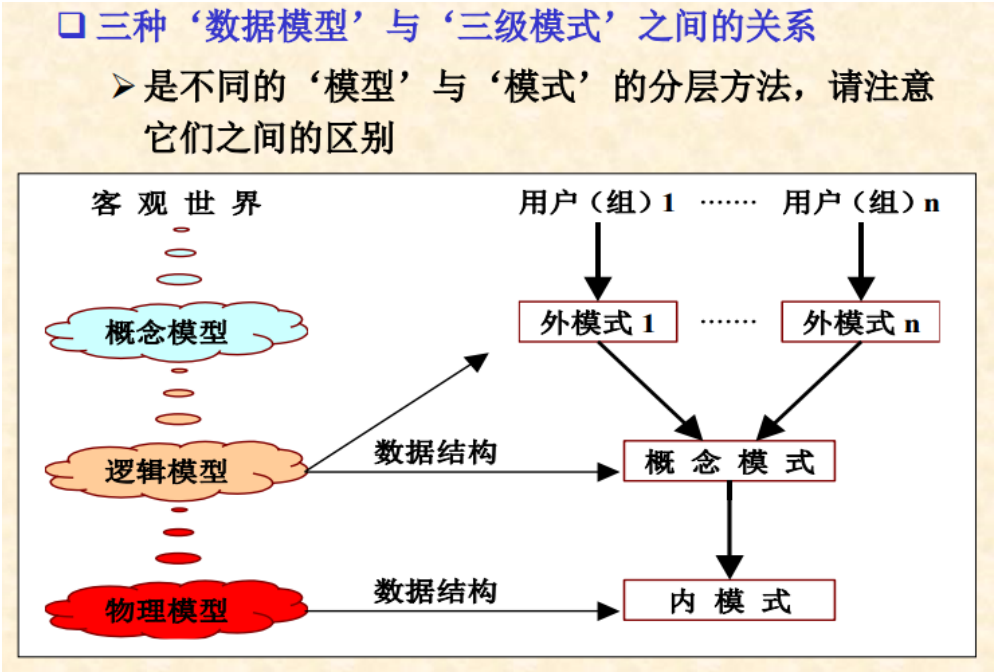
- 数据模式：是数据库系统中数据结构的一种表示形式，它具有不同的层次与结构方式
- 数据库的三级结构：三级模式、两级映射
  - 概念模式（简称模式）：是关于整个数据库中数据的全局逻辑结构的描述；数据类型，安全性，完整性，联系等
  - 外模式（用户模式、子模式）：是关于某个用户所需数据的逻辑结构的描述
  - 内模式（物理模式）：是关于数据库中数据的物理存储结构和物理存取方法的描述
  - 外模式-概念模式映射：外模式到概念模式的映射给出了外模式与概念模式的对应关系，可实现逻辑独立性；一个概念模式中可以定义多个外模式，而每个外模式是概念模式的一个基本**视图**
  - 概念模式-内模式映射：该映射给出了概念模式中数据的**全局逻辑结构**到数据的**物理存储结构**间的对应关系，可实现物理独立性



数据模型

2.1 数据模型的基本概念 *（概念）*

- 数据模型及其组成部分
  - 定义：描述数据的结构，定义在该数据结构上可以执行的操作以及数据之间必须满足的约束条件
  - 组成：**数据结构、数据约束、数据操作**
- 三种数据模型
  - 概念数据模型（概念模型）：侧重于对客观世界中复杂事物的结构描述及它们之间的内在联系的刻划，不涉及具体的描述细节和物理实现因素；ER模型、EER模型、面向对象模型...
  - 逻辑数据模型（数据模型）：着重于数据模型在数据库系统一级的实现，即利用具体的DBMS所提供的工具（DDL）来定义的数据模型
  - 物理数据模型（物理模型）：给出了数据模型在计算机内部的真正物理结构，是一种面向计算机物理实现的模型



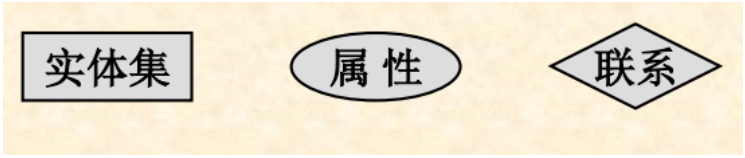
2.3 概念世界与概念模型 *（应用、概念）*

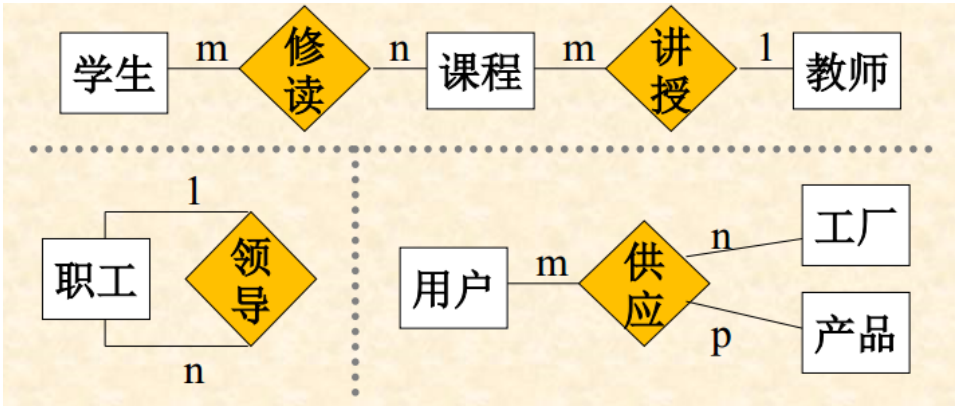
- E-R模型与E-R图：实体、属性、联系 ***（应用）***
  - 实体：客观存在且又能相互区别的事物
  - 属性：实体所具有的某种特性或特征；一个实体可以有多个属性
  - 联系：一个实体集中的实体与另一个实体集中的实体之间的对应关系；一对一、一对多、多对多；因联系的发生而产生的特性可以通过联系上的属性来表示

    ➢学习关系（学生，课程，*成绩*）

    ➢借阅关系（学生，图书，*借阅日期*，*归还如期*）

    ➢比赛（甲方，乙方，*比赛结果*）
  - E-R图

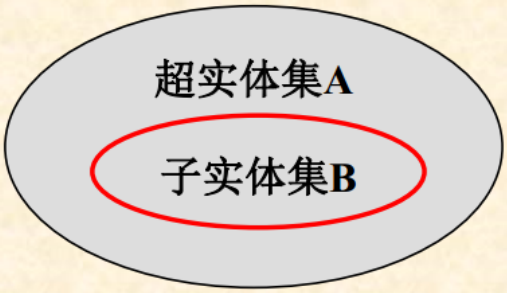
- EE-R模型与EE-R图：IS-A联系 *（概念）*
  - IS-A联系

    ❑IS-A联系

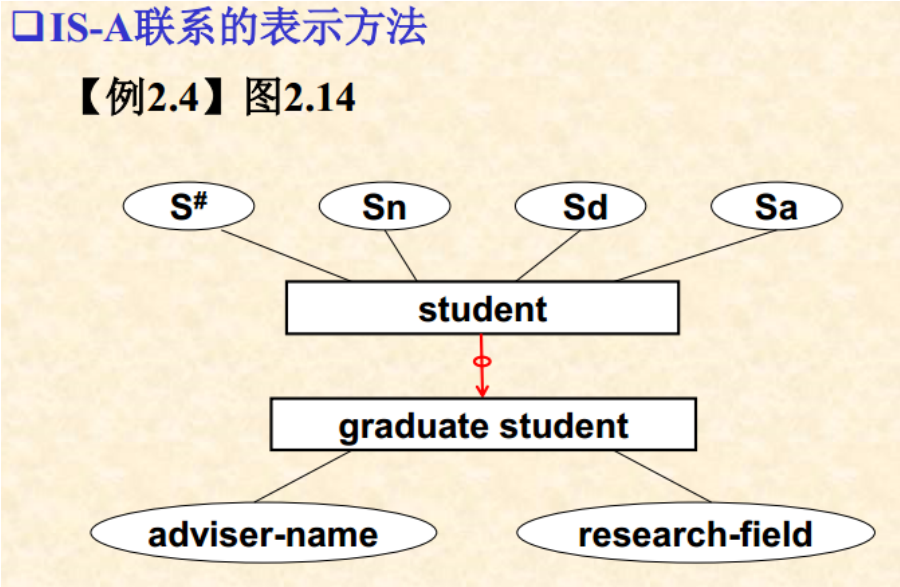
    ➢如果实体集B是实体集A的一个子集，且具有比实体集A更多的属性，则我们称在实体集A与实体集B之间存在着一种特殊的‘IS-A联系’。其中：

    –实体集A被称为实体集B的 超(实体)集

    –实体集B被称为实体集A的 子(实体)集

    ❑子集B可以通过IS-A联系继承超集A中的所有属性



- 弱实体：如果一个实体 A 的存在需要依赖于其他实体集中的某个实体的存在，那么实体 A 被成为弱实体，例如职工 vs 家属，学生 vs家长

2.4 信息世界与逻辑模型 *（概念）*

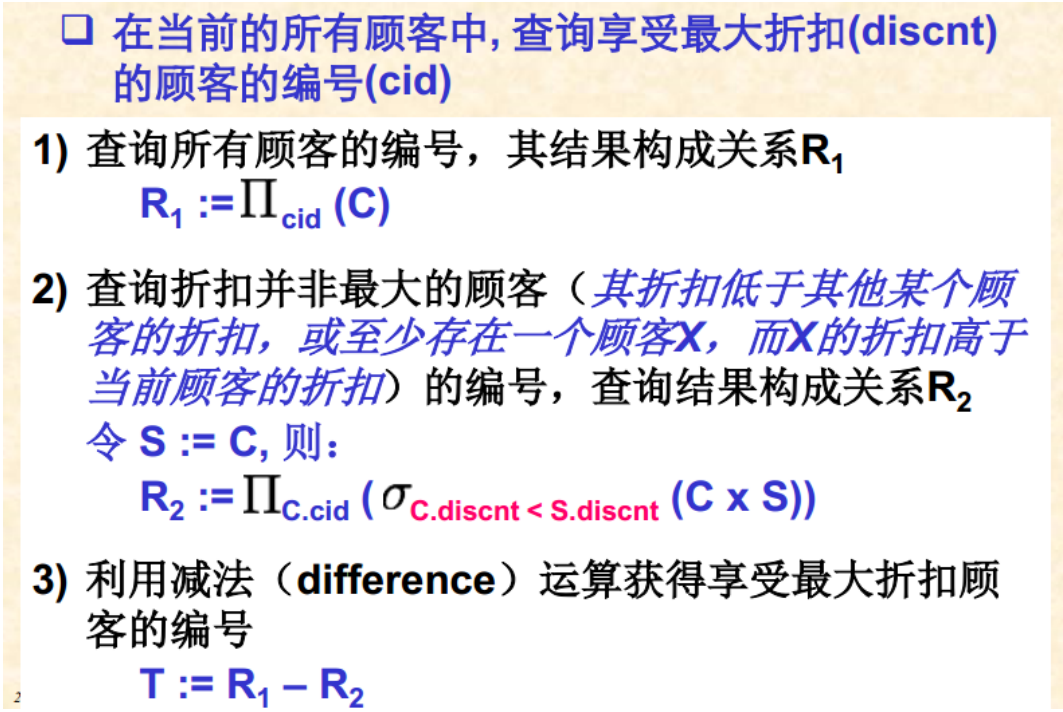
- 关系模型：关系，属性，值域，元组，关系数据库，关键字 *（概念）*
  - 关系：由行和列组成的二维表格
  - 元组：关系的每一行
  - 属性：关系的每一列
  - 值域
  - 关系数据库
  - 关键字：关系中的一个属性集的值能唯一标识关系中的一个元组，且又不含多余的属性值

关系数据库系统

3.3 关系代数 *（应用、概念）*

- 关系模型（概念）
  - 关系数据结构
    - 表结构（二维表的组成）
      - **表框架**：由n个命名的属性组成
        - n 被称为**表的元数**
        - 一个表框架可存放m个元组， m被称为**表的基数**
      - 元组：在表框架中可按行存放数据，其中的每行数据称为元组
    - **关系（二维表的性质）**：
      - **元组个数有限性**
      - **元组唯一性**
      - **元组次序无关性**
      - **元组分量的原子性**
      - **属性名唯一性**
      - **属性次序无关性**
      - **分量值域同一性**

        - 满足上述7个性质的二维表被称为**关系**，以符合上述条件的二维表为基本数据结构所建立的模型称为**关系模型**
    - 关键字
        - 候选键：在二维表中凡能唯一最小标识元组的属性集
        - 主键：被选中的候选键
        - 外键：如果表A中的属性集F是表B的键，则称该属性集F为表A的外键
    - 关系数据库：关系子模式-视图（view）
  - 关系操纵
    - 查询、插入、删除、修改
    - 空值的处理：主键不能为空；算术表达式->结果也为空；逻辑表达式->结果为假；统计...
  - **关系中的数据约束：实体完整性约束、参照完整性约束、用户定义的完整性约束**
  - 实体完整性约束：主键不为空
    - 参照完整性约束：外键要么取空值，要么是被引用表中当前存在的某元组上的主键值
    - 用户定义的完整性约束
- 关系代数
  - 关系的表示（概念）
    - 关系的表示：关系是元组的集合， 一个n元关系是一个n元有序组的集合
- 笛卡尔积
- 关系操纵的表示（应用）
  - 五种基本运算：选择、投影、笛卡尔积、并、差
    - 并：同类关系
    - 差：同类关系
    - 投影
    - 选择
    - 笛卡尔积



  - 关系代数中的扩充运算
    - 交：同类关系
    - 除：先投影，再除！！！
    - 联接：自然联接（同名属性只保留一份）、外联接、左外联接、右外联接

3.4 SQL*（应用）*

- **DDL数据定义**
  - 建表：create table 表名（列名 数据类型 【约束】）；
  - 修改：
    - 基表结构修改：alter table 表名 add 列名 数据类型; alter table 表名 drop 列名;
    - 约束修改：alter table 表名 add 【constraint 约束名】 约束（列名）；
  - 删除：drop table 表名
- **DML数据操纵**
  - 映像语句结构：

    目标：select ...（*、distinct、as)

    范围：from ... （别名 tablename 【aliasname】：自连接必须换名）

    条件：where ...

    分组：group by ...

    分组查询：having ...

    排序输出：order by ...

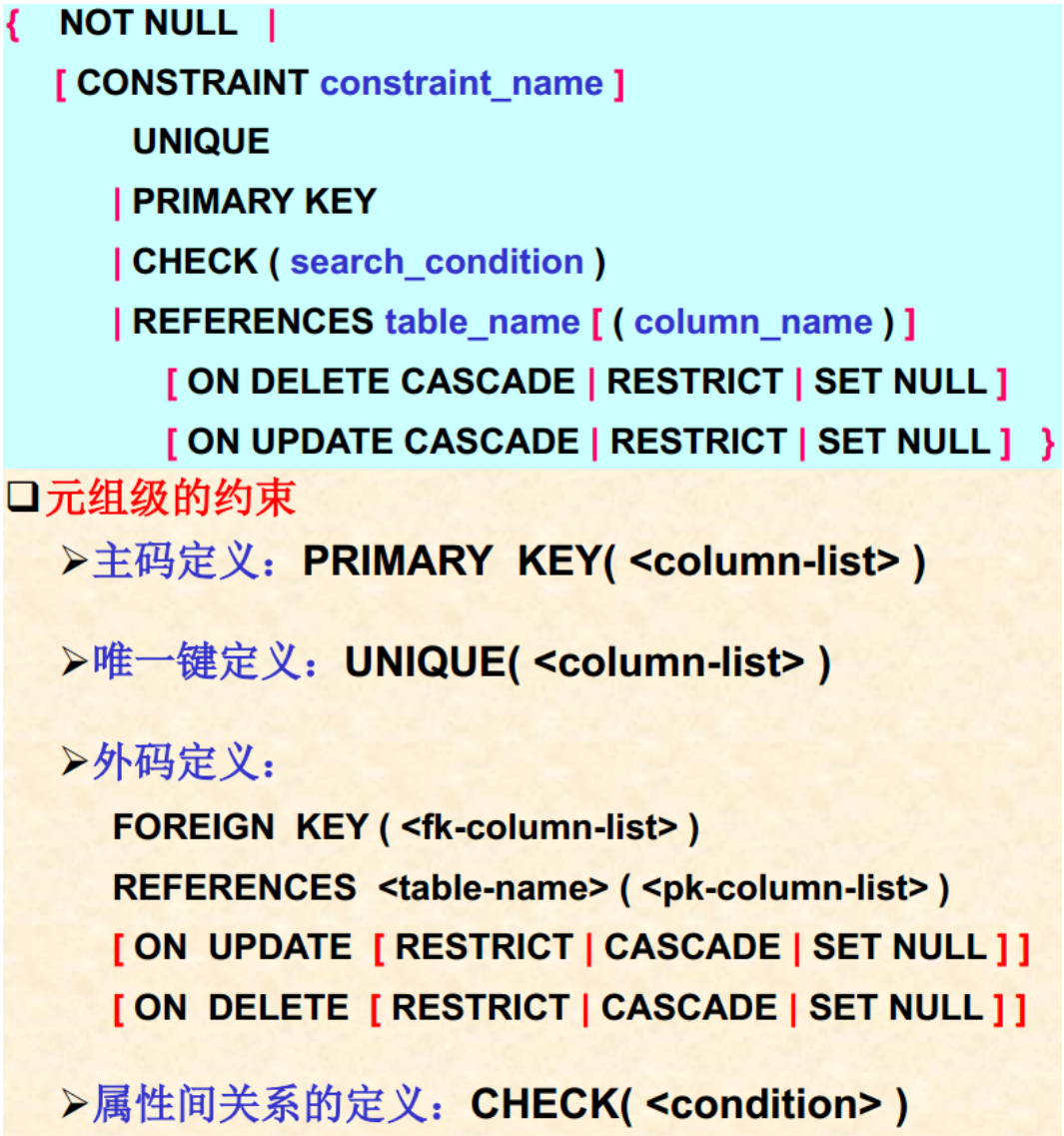
    limit ...
  - 基本查询功能：
    - 常用谓词：is null，is not null，distinct，between ... and ...，not between ... and ...，like，not like
    - like使用：column [NOT] LIKE val1 [ESCAPE val2]
      - _：匹配任一字符
      - %：匹配任意字符串
      - 转义字符val2之后的_或%表示自身
  - 嵌套查询：in（是否属于集合），some/any/all（单量和集合中元素的量化比较），exist（判断是否为空集）
  - 子查询的合并：UNION，INTERSECT，EXCEPT
  - 统计查询：count，sum，avg，max，min；不能在where子句中直接使用统计函数-->子查询
  - 除法：
- 更新
  - 删除：delete from tablename where ...；无where则删除全部
  - 插入：insert into tablename values (...) | subquery
  - 修改：update table set colname = ... where ...
- 视图：
  - 创建：create view viewname as ... [with check option]（with check option允许在视图上执行更新操作）
  - 删除：drop view viewname
  - 优点：提高了数据独立性；简化用户观点；提供自动的安全保护功能

数据库的安全性与完整性保护*（应用）*

4.1 数据库的安全性保护

- SQL 语言所提供的与数据库安全保护有关的命令（应用）
  - 授权：GRANT <操作权限列表> ON <操作对象> TO <用户名列表> [WITH GRANT OPTION]（with grant option:拿到权限的用户能否再授权）
    - e.g. grant SELECT,UPDATE on S to XULIN with grant option；grant UPDATE (G) on SC to XULIN
  - 回收：REVOKE <操作权限列表> ON <操作对象> FROM <用户名列表> [RESTRICT | CASCADE]
    - CASCADE：连锁回收（e.g. xulin又授权给a，则一起回收）
    - RESTRICT：不存在连锁回收时才回收，否则不回收

4.2 数据库的完整性保护

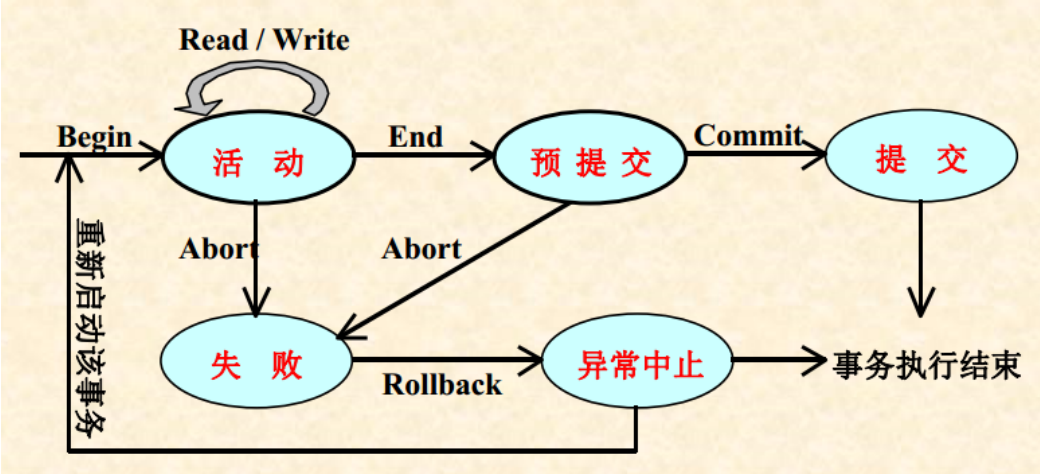
- 完整性规则的三个内容：实体完整性，参照完整性，用户定义的完整性
- 完整性约束的设置、检查与处理：
  - 对约束命名：CONSTRAINT <约束名> <完整性约束定义子句>
  - 

  - 外键：on xx cascade，restrict，set null，no action（restrict：操作前检查，no action：操作后检查，仍符合则不报错）

事务处理、并发控制与故障恢复技术

5.1 事务处理 *（概念）*

- 事务的定义与 ACID 性质
  - 事务：由某个用户定义的一个不能被打断的数据库操作序列（可以是一条、一堆sql语句、整个程序...）；是恢复和并发控制的基本单位
  - **ACID性质**
    - **原子性Atomicity**：事务中的操作要么都做要么都不做
    - **一致性Consistency**：一个事务的成功执行总是将数据库从一个一致的状态转换到另一个一致的状态（正确性：满足显式完整性约束、用户心目中隐式约束）
    - **隔离性Isolation**：一个事务的执行与并发执行的其它事务之间是相互独立的，互不干扰；可串行化
    - **持久性Durability**：一个事务一旦完成其全部操作后，它对数据库的所有更新应永久地反映在数据库中，即使以后系统发生故障也应该能够通过故障恢复来保留这个事务的执行结果
- 事务活动及其状态转换图



- 事务控制及相关的参数设置语句?
  - 事务的提交和回滚：commit、rollback
  - 事务的读写类型：SET TRANSACTION READONLY|READWRITE
  - 隔离级别：SET TRANSACTION ISOLATION LEVEL READUNCOMMITTED|READCOMMITTED|READREPEATABLE|SERIALIZABLE

5.2 并发控制技术 *（概念）*

- 事务
  - 事务的并发性，并发控制
  - 调度、串行、可串行化调度、冲突与冲突可串行化、冲突可串行化的判定方法
    - 调度：一个或多个事务中的数据库访问操作，按照这些操作被执行的时间排序所形成的一个操作序列
    - 串行：一个事务所有操作结束，另一个事务所有操作以此类推
  - 可串行化：多个事务并发执行的最终结果，应该与它们的某种串行执行的最终结果相等
    - 冲突：调度中一对连续操作，交换它们的执行顺序，涉及的事务中至少有一个的行为会改变
    - 冲突等价：如果通过一系列相邻操作的非冲突交换能够将一个调度转换为另一个调度，则我们称这两个调度是冲突等价的
    - 冲突可串行化：如果一个调度 S 冲突等价于一个串行调度，则我们称调度 S 是“冲突可串行化” 的；冲突可串行化调度一定是可串行化调度，可串行化调度不一定是冲突可串行化调度

        - 冲突可串行化的判定
    - 三种数据不一致现象
        - 丢失修改（写-写）：T1、T2同时读并修改，T2提交覆盖T1提交结果
        - 脏读：T1修改并写会磁盘后，T2读，T1撤销，T1修改的恢复原值，则T2读到的是脏数据
        - 不可重复读（读-更新【修改、插入、删除】）：T1读数据后，T2更新，T1无法再现前一次读的结果
- 封锁
    - 共享锁、排它锁、锁相容矩阵、合适事务
        - 排它锁（X锁）：只允许T读取和修改A，其他事务不能对A加锁
        - 共享锁（S锁）：T可读A不能改A，其他事务也只能加S锁
        - 锁相容矩阵

		其它事务已持有的锁		
		X锁	S锁	-
当前事务申请的锁	X锁	No	No	Yes
	S锁	No	Yes	Yes

        - 合适事务：如果一个事务在访问数据库中的数据对象A之前按照要求申请对A的封锁，在操作结束后释放A上的封锁，这种事务被称为合适事务；是保证并发事务的正确执行的基本条件
    - 基于封锁技术的并发控制实现方法
    - 封锁协议：三级封锁协议、两阶段封锁协议
        - 三级封锁协议：
            - 一级封锁协议：修改前加X锁直到事务结束；防止丢失修改
            - 二级封锁协议：一级+读前S锁，读完释放；防止丢失修改、脏读，不能保证可重复读
            - 三级封锁协议：一级+读前S锁直到事务结束；防止丢失修改、脏读、不可重复读
        - 两阶段封锁协议：
            - **第一个阶段：申请并获得封锁**
                - **在此阶段中，事务可以申请其整个执行过程中所需要的锁，此阶段也可称为‘扩展阶段’**
            - **第二阶段：释放所有申请获得的锁**
                - **此阶段也可称为‘收缩阶段’**
                - **事务一旦开始释放封锁，那么就不能再申请任何封锁**
    - 合法调度：两阶段封锁协议与冲突可串行化的关系

- 在每一个事务T_i中
  - 第1点：采用如下的封锁协议：
    - 读动作 $r_i(A)$ 之前必须有 $sl_i(A)$ 或 $xl_i(A)$，而且在两者之间没有 $u_i(A)$
    - 写动作 $w_i(A)$ 之前必须有 $xl_i(A)$，而且在两者之间没有 $u_i(A)$
    - 每一个 $sl_i(A)$ 或 $xl_i(A)$ 之后必须有一个 $u_i(A)$
  - 第2点：必须遵循‘两阶段封锁协议’
    - 在任何 $sl_i(A)$ 或 $xl_i(A)$ 之前不能有 $u_i(B)$
      - A和B可以是同一个数据对象
- 第3点：保证事务调度的合法性
  - 对任意两个不同的事务 T_i 和 T_j，其调度必须满足：
    - 如果 $xl_i(A)$ 出现在调度中，那么后面不能再有 $sl_j(A)$ 或 $xl_j(A)$，除非中间插入了 $u_i(A)$
    - 如果 $sl_i(A)$ 出现在调度中，那么后面不能再有 $xl_j(A)$，除非中间插入了 $u_i(A)$

- 多粒度封锁：封锁粒度与多粒度封锁
  - 封锁粒度：一把锁可以封锁的数据对象的大小；锁的封锁对象可以是数据库中的逻辑数据单元，也可以是物理数据单元
  - 多粒度封锁：同时支持多种封锁粒度供事务选择使用
    - 通过选择合适的‘封锁粒度’来达到如下目的
      - 通过加大‘封锁粒度’来减少‘锁’的数量，降低并发控制的开销
      - 通过降低‘封锁粒度’来缩小一把锁可以封锁的数据范围，减少封锁冲突现象，提高系统并发度
    - 意向锁：如果一个结点加意向锁，则说明该结点的**下层结点**正在被加锁；对任一结点加锁时，必须先对它的上层结点加意向锁
    - 意向共享锁（IS锁）：如果对结点N加‘IS锁’，表示准备在结点N的某些后裔结点上加‘S锁’
    - 意向排它锁（IX锁）：如果对结点N加‘IX锁’，表示准备在结点N的某些后裔结点上加‘X锁’
    - 共享意向排它锁（SIX锁）：如果对结点N加‘SIX锁’，表示对结点N本身加‘S锁’，并准备在N的某些后裔结点上加‘X锁’（想读一个对象，并更新后代节点）

❑ 带有意向锁的锁相容矩阵

		其它事务已持有的锁				
		S锁	X锁	IS锁	IX锁	SIX锁
当前事务申请的锁	S锁	Yes	No	Yes	No	No
	X锁	No	No	No	No	No
	IS锁	Yes	No	Yes	Yes	Yes
	IX锁	No	No	Yes	Yes	No
	SIX锁	No	No	Yes	No	No

Yes：表示相容的请求　　No：表示不相容的请求

5.3 数据库恢复技术 *（概念）*

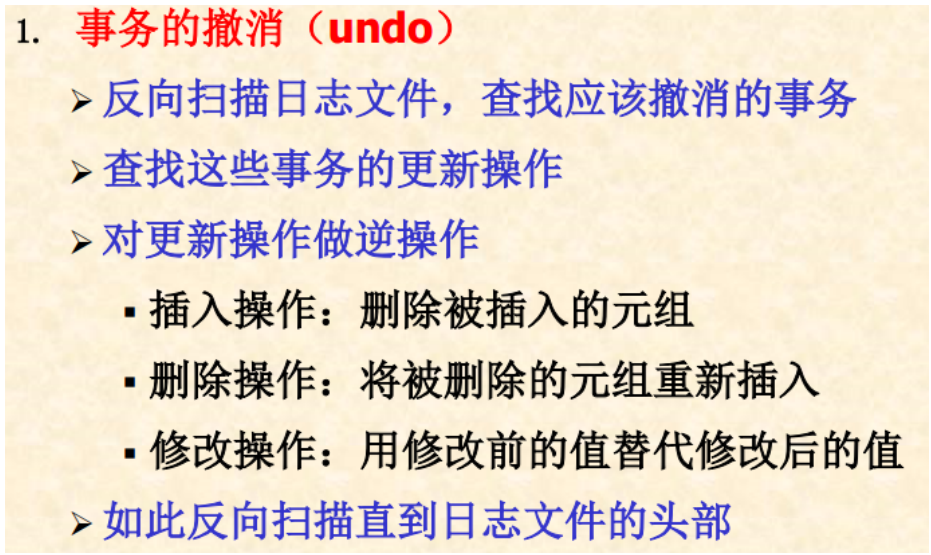
- 数据库恢复的含义、方法、常用措施
  - 含义：在数据库遭受破坏后及时进行恢复的功能
  - 方法：利用数据冗余原理，将数据库中的数据在不同的存储介质上进行**冗余**存储，当数据库本身收到破坏时，可以利用这些冗余信息进行恢复
  - 措施：数据转储、日志、数据库镜像
- 数据库故障的分类
  - 小型故障：事务内部故障；故障的影响范围在一个事务之内，不影响整个系统的正常运行-->UNDO
  - 中型故障：可导致整个系统停止工作，但磁盘数据不受影响。在系统重启时，可通过当前的日志文件进行恢复
    - 系统故障（CPU/OS/DBMS故障，断电...）-->UNDO所有未完成，REDO所有已提交（可能还在缓冲区，未写入数据库）
    - 外部影响
  - 大型故障：磁盘故障、病毒、黑客；可导致内存及磁盘数据的严重破坏，需要对数据库做彻底的恢复
- 数据库故障恢复三大技术
  - 数据转储：定期地将数据库中的内容复制到其它存储设备中去的过程
    - 静态（转储时无事务运行）、动态（转储与事务并发进行；转储时建立日志文件）
    - 海量、增量
  - 日志：是由数据库系统创建和维护的，用于自动记载数据库中修改型操作的数据更新情况的文件
    - 内容：

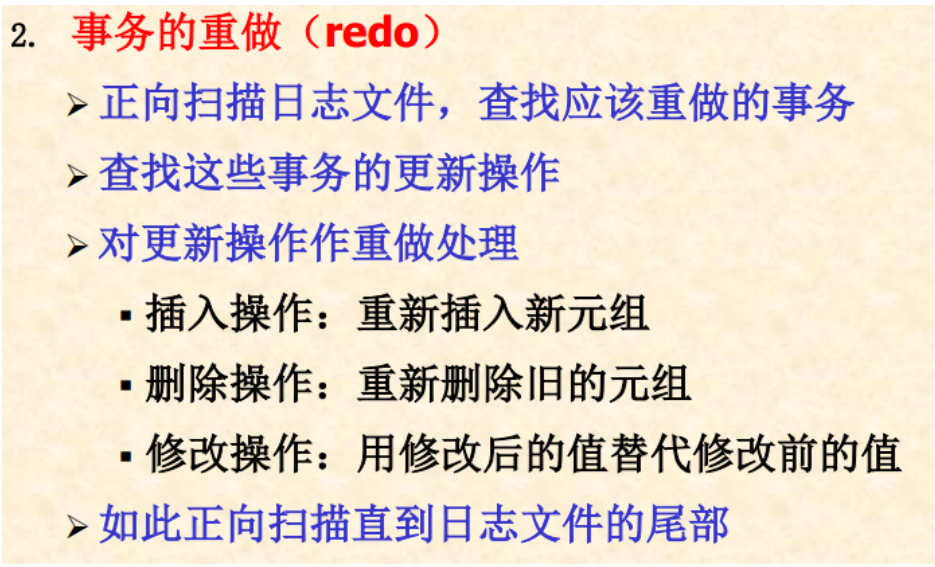
      ➢ 每个更新操作的事务标识、更新对象、更新前的值 和/或 更新后的值

      ➢ 每个事务的开始、结束等执行情况

      ➢ 其它信息
    - 组成：日志是'日志记录' 的一个序列；由于事务通常是并发执行的，所以多个事务的日志记录通常是交错在一起的

        - 作用：
            - 确保事务执行的原子性
            - 实现增量转储
            - 实现故障恢复
        - 记载原则：按照操作执行的先后次序，遵循**先写日志，后修改数据库**的原则
        - 三种类型：UNDO日志（用于被放弃事务（包括在发生故障时尚未结束的事务）的撤消工作）、REDO日志（用于已提交事务的重做工作）、UNDO/REDO日志
    - 事务的撤销undo与重做redo





- 恢复策略
    - 小型故障：UNDO
    - 中型故障：未完成反向UNDO，已完成正向REDO
    - 大型故障：

- 先利用后备副本进行数据库恢复，再利用日志进行数据库的恢复。具体步骤如下
  1. 将后备副本中的数据拷贝到数据库中
  2. 检查日志文件：确定哪些事务已经执行结束，哪些尚未结束
  3. 按照日志的记载顺序
     a) 逆向：对尚未结束的事务作撤消处理(undo)
     b) 正向：对已经结束的事务作重做处理(redo)

数据库的物理组织

7.6 索引技术与散列技术

- 顺序文件的组织方式
- 索引文件的组织方式
  - **在顺序文件上的索引技术*（应用）***：稠密索引、稀疏索引、多级索引
    - 稠密索引：数据文件中的每条记录在索引文件中都存在一个相对应的索引项（二分查找+1）
    - 稀疏索引：顺序文件，在索引文件中只为数据文件的每个磁盘块设一个索引项，记录该磁盘块中第一条数据记录的关键字值及该磁盘块的首地址（二分查找+1）
    - 多级索引：在索引文件上再建立索引，索引文件本身是顺序文件，因此是稀疏索引
  - 非顺序文件中的索引技术
- B/B+树文件
- Hash文件

关系数据库规范化理论

8.1 概述*（概念）*

- 模式设计质量的评价指标：数据冗余度，有无插入/更新等更新异常

8.2 规范化理论*（概念/应用）*

- 函数依赖FD：完全/部分、平凡/非平凡、直接/传递
  - 函数依赖：设有关系模式R(U)， U是关系模式R的属性集合，X、 Y是U的子集。若对于任一个符合关系模式R(U)的关系 r 中的**任一元组 t 在X中的属性值确定后，则元组 t 在Y中的属性值也必确定**，则称Y函数依赖于X， 或者称X函数决定Y， 并记为X→Y；X为决定因素，Y为依赖因素；若R（U）上存在函数依赖X->Y，任取两个元组t1和t2，若t1[X]=t2[X],则t1[Y]=t2[Y]
  - 平凡/非平凡：X->Y，Y不是X的子集--非平凡；否则平凡，如（A，B）->A
  - 完全/部分：X→Y， 且对任何X的真子集X' 都没有X'→Y， 则称Y完全函数依赖于X

- 传递/非传递（直接）：

  ➢在关系模式R(U)中，如有X⊆U，Y⊆U，Z⊆U且满足：X→Y，Y⊄X，Y↛X，Y→Z，则称Z传递函数依赖于X；否则，称为非传递函数依赖（*直接函数依赖*）。

- Armstrong公理系统：
  - 自反规则：如果Y是X的子集，则： X → Y
  - 增广规则：如果X → Y， 则： XZ → YZ
  - 传递规则：如果X → Y， Y → Z， 则： X → Z
  - 分解规则：如果X → YZ， 则： X → Y （自反YZ->Y，传递）
  - 合并规则：如果 X → Y 且 X → Z， 则： X → YZ （增广X->XY，XY->YZ，传递）
  - 伪传递规则：如果X → Y且WY → Z， 则： WX → Z （增广，传递）
- 基于函数依赖的关键字定义*（概念）*：

  ➢在关系模式 R(U, F) 中，如有 K⊆U 且满足：

  $K \xrightarrow{f} U$

  则称 K 为 R 的关键字。

- **属性集闭包的计算算法*（应用）***：由被 F 逻辑蕴涵（由armstrong推出）的所有函数依赖关系构成的集合被称为函数依赖集 F 的闭包，并记为 F+

- 与函数依赖有关的范式
  - 范式：
    - 1NF：不可分割
    - 2NF：1NF+每个非主属性都完全函数依赖于关键字
    - 3NF：2NF+每个非主属性都不传递函数依赖于关键字
    - BCNF：1NF+若 X->Y 则 X 必含有该关系模式的关键字（注意：包含关键字而不是主属性）