亿图脑图 MindMaster

数据库

绪论

- 数据库系统概述
  - 数据库的四个基本概念：数据、数据库、数据库管理系统、数据库系统
  - 数据管理技术的产生和发展
    - 人工管理阶段、文件系统阶段、数据库系统阶段
    - 数据是否保存、谁管理数据、数据是否共享、数据独立性与共享性
  - 数据库系统的特点
    - 数据结构化：数据库实现整体数据的结构化。这也是数据库系统与文件系统的本质区别，文件中的记录内部具有结构，但记录的结构与记录之间的联系被固化在程序中，需要程序员加以维护。
    - 整体结构化是指数据库中的数据不再仅仅针对某一个应用，而是面向整个组织或企业，不仅数据内部是结构化的，而且整体是结构化的，数据之间是具有联系的
    - 数据之间共享性高、冗余度低且易扩充：数据共享可以大大减少冗余，节约存储空间，还可以避免数据之间的不相容性和不一致性。
    - 数据独立性高：物理独立性和逻辑独立性
    - 数据由数据库管理系统统一管理和控制：安全性保护、完整性检查、并发控制、数据库恢复
- 数据模型：对现实世界数据特征的抽象
  - 概念模型（信息模型）：按用户的观点来对数据和信息建模，主要用于数据库设计
    - 涉及概念：实体、属性、码、实体型、实体集、联系
    - 实体之间的联系有一对一、一对多、多对多等多种类型
  - 逻辑模型和物理模型
    - 逻辑模型：包含层次模型、网状模型、关系模型等，是按计算机系统的观点来对数据建模，主要用于数据库管理系统的实现
      - 层次模型：有且只有一个结点没有双亲，其他任何一个结点只有一个双亲结点。层次模型结构清楚，查询效率高，但这种联系是一对多的联系，该数据库也只能处理一对多的联系，现实生活中有很多多对多的关系不适用于层次模型；查询子女结点必须通过双亲节点，也不是很方便。
      - 网状模型：允许多个结点没有双亲，一个结点可以有多个双亲。网状模型可以更直接地表述现实世界，但其结构比较复杂，用户不易掌握。
      - 关系模型：就是一张不可再分的二维表。关系模型概念单一，无论实体还是实体之间的联系都用关系表；关系模型的存取路径对用户透明。
    - 物理模型：是对数据最底层的抽象，描述数据在系统内部的表示方式和存取方法，或在磁盘或磁带上的存储方式和存取方法，是面向计算机系统的。物理模型的具体实现是数据库管理系统的任务。
- 数据库系统的结构
  - 三级模式结构
    - 模式也称逻辑模式，是数据库中全体数据的逻辑结构和特征的描述，仅涉及型的描述，不涉及具体的值。一个模式可以有多个实例。模式是相对稳定的，实例是经常变动的。一个数据库只有一个模式。
    - 外模式也称子模式或者用户模式，它是数据库用户能够看见的和使用的局部数据的逻辑结构和特征的描述，是数据库用户的数据视图，是与某一应用有关的数据的逻辑表示。外模式通常是模式的子集，一个模式可以有多个外模式。
    - 内模式也称存储模式，一个数据库只有一个内模式，它是数据物理结构和存储方式的描述，是数据在数据库内部的组织方式。如，记录的存储方式是升序还是降序，索引是用B+树索引还是哈希索引。
  - 两级映像
    - 外模式-模式映像：外模式/模式映像定义了该外模式与模式的对应关系。当模式改变时，由数据库管理员对各个外模式/模式映像作相应改变，可以使外模式保持不变。应用程序是根据外模式编写的，从而应用程序不用改变，保护了数据与程序的逻辑独立性。
    - 模式-内模式映像：数据库中只有一个模式，也只有一个内模式。当模式改变时，由数据库管理员对各个模式/内模式映像作相应改变，可以使模式保持不变，从而应用程序不用改变，保护了数据与程序的物理独立性。
- 思考：
  1、试述数据、数据库、数据库管理系统、数据库系统的概念
  2、使用数据库系统有什么好处
  3、试述文件系统和数据库系统的区别和联系
  4、举出适合用文件系统而不是数据库系统的应用例子，以及适合用数据库系统的应用例子
  5、试述数据库系统的特点
  6、数据库管理系统的主要功能是什么
  7、什么是概念模型，试述概念模型的作用
  8、试述关系数据库的特点

关系数据库

- 域、笛卡尔积、主码、全码、候选码
- 关系操作：
  5个基本操作：选择、投影、并、差、笛卡尔积
  其余的：连接、除、交
- 思考：
  1、试述关系模型的三个组成部分
  2、简述关系数据语言的特点和分类
  3、举例说明关系模式和关系的区别
  4、试述关系模式的完整性规则。在参照完整性中，什么情况下外码属性的值可以为空值
  5、试述等值连接和自然连接的区别和联系
  6、关系代数的基本运算有哪些？如何用这些基本运算来表示其他运算？

SQL的相关使用

- 模式、表、视图、索引
  创建CREATE、删除DROP、修改ALTER
  其中，模式和视图是不允许ALTER的
- 数据字典是关系数据库管理系统内部的一组系统表，它记录了数据库中所有的定义信息，包括关系模式定义、视图定义、索引定义、完整性约束定义等。在进行查询优化和查询处理时，数据字典中的信息是其重要依据。
- 聚集函数遇到空值时，除了COUNT(*)外，都跳过空值而处理非空值
- GROUP按某列关系分组
  HAVING短语用于筛选
  ORDERED指按某列关键字排序
- 行列子集视图：从单个基本表导出，并且只是去掉了基本表的某些行和某些列，但保留了主码，则为行列子集视图。
  带表达式的视图：带虚拟列的视图

  视图消解：从数据字典中取出视图的定义，把定义中的子查询和用户的查询结合起来，转换成对等价的基本表的查询。

  行列子集视图是可更新的。

  不允许更新（插入INSERT、删除DELETE、修改UPDATE）的情况：
  1、视图由两个以上基本表导出，则该视图不允许更新。
  2、若视图的字段来自字段表达式或常数，则不允许对此视图执行INSERT或UPDATE操作，但可以DELETE
  3、若视图的字段来自聚集函数，则该视图不允许更新
  4、若视图定义中含有GROUP BY子句，则该视图不允许更新
  5、若该视图含有DISTINCT短语，则该视图不允许更新。

  视图的作用：
  1、简化用户使用，使用户关注自己需要的数据
  2、使用户多角度地看待同一数据
  3、对重构数据库提供了一定程度上的逻辑独立性（模式-外模式映像）
  4、对机密数据提供安全保护

数据库安全性

- 数据库中的不安全因素
  - 指保护数据库以防止不合法使用所造成的数据泄露、更改或破坏
  - 非授权用户对数据库的恶意存取和破坏
  - 数据库中重要或敏感的数据被泄露
  - 安全环境的脆弱性
- 安全标准简介
- 用户身份鉴别：是数据库管理系统提供的最外层安全保护措施。每个用户在系统中都有一个用户标识（用户名+用户标识号）。每次用户进入系统时由系统进行核对，通过鉴定后才提供使用数据库管理系统的权限。
  - 静态口令鉴别
  - 动态口令鉴别
  - 生物特征识别
  - 智能卡鉴别
- 存取控制
  - 自主存取控制DAC：由于自主存取控制中用户自由决定数据的存取权限，可能存在数据的无意泄露。核心原因：这种机制仅仅通过对数据的存取权限进行安全控制，而数据本身并无安全性标记。
  - 强制存取控制：数据库管理系统所管理的实体被分为主体和客体两大类。主体是系统中的活动实体，既包括数据库管理系统所管理的实际用户，也包括代表用户的各进程。客体是系统中的被动实体，是受主体操纵的。
  - 规则1：仅当主体的密级>=客体的密级时，主体才能读取相应的客体。
    规则2：仅当主体的密级<=客体的密级时，主体才能够写相应的客体。（主要用来防内鬼，若违反这一规则，TS密级的主体恶意地将TS密级的客体降为（写）P密级，这样大家就都能看到了，造成了TS密级数据的泄露。
  - 视图机制：通过视图机制把无权存取的用户隐藏起来
  - 审计：将用户对数据库的所有操作自动记录下来放入审计日志。审计员可以利用审计日志监控数据库中的各种行为，重现导致数据库现有状况的事件，找出非法存取数据的人、时间、内容等。
  - 数据加密：存储加密、传输加密
- 思考：
  1、什么是数据库的安全性
  2、举例说明对数据库安全性产生威胁的因素
  3、试述实现数据库安全性控制的常用方法和技术
  4、什么是数据库中的自主存取控制方法和强制存取控制方法
  5、解释强制存取控制机制中主体、客体、敏感度标记的含义
  6、举例说明强制存取控制机制是如何确定主体能否存取客体的
  7、什么是数据库的审计功能，为什么要提供审计功能？

数据库完整性

- 实体完整性
  - 指数据的正确性和相容性。数据的正确性指数据是符合现实世界语义、反映当前实际状况的；数据的相容性是指数据库同一对象在不同关系表中的数据是符合逻辑的。
  - 具体表现：唯一标识记录的属性不能为空
  - 定义：1、若主码只有一个属性，则既可列级约束条件（Sno char(9) PRIMARY KEY），也可表级约束条件
    2、若主码有多个属性，则只能表级约束条件(PRIMARY KEY(Sno,Cno))
  - 实体完整性检查和违约处理：在插入记录或者更新记录时，需根据实体完整性规则进行检查：
    1、检查主码值是否唯一，不唯一则拒绝插入或修改
    2、检查主码的各个属性是否为空（主码可能有多个属性，若在学生选课关系中，学生-课程唯一标识一条记录），只要有一个为空就拒绝插入或修改
- 参照完整性
  - 具体表现：学生所在的专业需要是学校已有的专业，学生选的课必须是课程表中已有的课。
  - 参照完整性检查和违约处理：
    1、在参照表中插入元组或修改外码值，可能破坏参照完整性
    2、在被参照表中删除元组或修改主码值，可能破坏参照完整性
  - 处理方法：
    1、拒绝执行（默认策略）
    2、级联操作，当删除或修改被参照表的一个元组导致与参照表的不一致时，删除或修改参照表中所有导致不一样的元组（对专业改名等，很有效）
    3、设置为空值：当删除或修改被参照表的一个元组时造成了不一致，则将参照表中所有不一致的元组的对应属性设置为空值（表示所有学生专业等待分配）
  - 注意：外码有时可以为空，因为这可以表示学生尚未被分配专业；有时不能为空，比如在学生选课数据表中，学生不可能选一个课程号为空的课。因此，在定义外码时需要说明是否可以为空。
- 用户自定义完整性：针对某一具体应用的数据必须满足的语义需求
  - 属性上约束条件的定义：列值为空、列值唯一、用CHECK短语指定列值应该满足的条件
  - 属性上约束条件的检查和违约处理：不满足则操作拒绝执行
- 完整性约束命名子句：上述提到的三类完整性语句都是在CREATE TABLE语句中定义的，不便于删除。因此提供完整性约束命名子句，给完整性条件命名，方便灵活增加、删除一个完整性约束条件
  - 完整性约束命名子句：
    Sname CHAR(20)
    CONSTRAINT C1 NOT NULL
  - 修改：
    DROP CONSTRAINT C1
    ADD CONSTRAINT C1 CHECK...
- 断言:通过声明断言，来指定更具一般性的约束。任何使断言不为真的操作都会被拒绝执行。换言之，“断言”数据库课程最多60人选修，就不能超过这个数。更像是未卜先知的法则。
  - CREATE ASSERTION <断言名> <CHECK子句>
  - DROP ASSERTION <断言名>
- 触发器：用户定义在关系表上的一类由事件驱动的特殊过程，一旦定义，任何用户对表的增、删、改、查操作均由服务器自动激活相应的触发器。触发器又叫“事件-条件-动作”规则，即发生某个事件时，如果xx，就xxx。（违约处理）
  - 只有创建表的用户才能在表上建触发器
  - 触发器只能定义在基本表上，不能定义在视图上
- 思考：
  1、什么是数据库的完整性
  2、数据库的完整性概念和数据库的安全性概念有什么区别和联系
  3、什么是数据库的完整性约束条件？
  4、关系数据库管理系统的完整性控制机制应具有哪三方面的功能
  5、关系数据库在实现参照完整性时需考虑哪些方面？
  6、在关系数据库中，当操作违法实体完整性、参照完整性和用户自定义完整性约束条件时，一般是如何分别进行处理的？

关系数据理论

- 函数依赖：X函数确定Y，Y函数依赖于X
- 非平凡的函数依赖：(Sno,Cno)->Grade
  平凡的函数依赖：(Sno,Cno)->Sno。没什么出彩的，当然自己能推出自己
  完全函数依赖：(Sno,Cno)->Grade。tm的少一个都推不出来
  部分函数依赖：(Sno,Sname)->Sdept。只要一个Sno就可以推出来所在系所了
  传递函数依赖：Sno->Sdept->Sin
- 超码：可以区分记录的一个属性或多个属性的集合(候选码+非主属性)
  候选码：超码的最小集（Sno）
  主码：从候选码里选一个
- 规范化：一个低一级范式的关系模式通过模式分解可以转换为若干个高一级范式的关系模式的集合，这种过程就叫做规范化
  第一范式1NF：每个分量都是不可再分的数据项（表中没有表）
  第二范式2NF：在第一范式的基础上，消除部分函数依赖。
  第三范式3NF：在第二范式的基础上，消除传递函数依赖
  BCNF：在第三范式的基础上，无其他决定因素。
- 若存在部分函数依赖，则会产生很多问题。如学生成绩信息表，有Sno、Sdept、Mname、Cno、Grade中，主码是(Sno，Cno)，存在(Sno,Cno)完全函数决定Grade，也存在(Sno,Cno)部分函数决定Sdept等字段，因为有Sno就够了。
  （可以理解为绑定关系）

  数据冗余，因为大量的冗余数据，系主任没必要存这么多
  插入异常，若一个系刚成立，尚无学生，则无法录入系主任信息
  更新异常，更换系主任时要更换所有元组
  删除异常，若某系学生全部毕业，则删除学生成绩的同时，系主任信息也会被删除
- 若存在传递依赖，也会产生问题。如学生信息表Sno,Sdept,Mname。
  所在系和系主任的对应关系没必要存这么多，又可以拆分。

数据库设计

- 需求分析阶段：准确了解客户需求
  概念结构设计阶段：通过对用户需求进行综合、归纳和抽象，形成一个独立于具体数据库管理系统的概念模型
  逻辑结构设计阶段：将概念结构转换为某个数据库管理系统所支持的数据模型，并对其进行优化
  物理结构设计阶段：为逻辑数据模型选取一个最适合应用环境的物理结构（包括存储结构和存取方法）
  数据库实施阶段
  数据库运行和维护阶段
- 概念模型的特点：能真实、客观地反映现实世界，包括事物和事物之间的联系，是现实世界的一个真实模型，易于理解，易于更改，易于向关系、网状、层状等各种数据模型转换。
- E-R图相关（概念结构设计）：
  1、实体之间的联系：两个实体型之间的联系（一对一、一对多、多对多）、两个以上的实体型联系（一对一、一对多、多对多）、单个实体型内的联系（一对一、一对多、多对多，如职工实体型中存在领导和被领导的关系，职工干部领导若干职工）
  2、实体型用矩形表示，属性用椭圆形表示，联系用菱形表示。（如果一个联系具有属性，则这些属性也要用无向边与该联系连接起来，如供应商、项目与零件的供应关系，彼此都是多对多的关系，但对于供应关系，也有供应量的属性。
  3、实体与属性的划分原则：
  （1）作为属性，不能有需要再被描述的性质：职称若不与工资、福利等挂钩，则作为属性合适，若挂钩则作为实体合适。
  （2）属性不能和其他实体产生联系：每个病人都有一个病房号，若无医生实体，则作为属性合适，但一个医生会负责几个病房的病人医疗工作，此时病房又作为实体合适。
  4、E-R图的集成：
  （1）合并E-R图，生成初步E-R图
  属性冲突：属性域冲突、属性取值单位冲突
  命名冲突：同名异义、一义多名
  结构冲突：同一对象在不同的应用里有不同的抽象（职工在某一局部应用中被当做属性，在另一局部应用中被当做实体）、同一实体在不同子系统的E-R图中所包含的属性个数和属性排列次序不完全相同、实体间的联系在不同的E-R图中是不同的类型。
  （2）消除不必要的冗余，设计基本E-R图
  冗余的数据指可由基本数据导出的数据，冗余的联系指可由其他联系导出的联系。冗余数据和冗余联系极易破坏数据完整性，给数据库维护增加困难
- 逻辑结构设计：
  1、E-R图向关系模型的转变：
  一个实体型转换为一个关系模式，关系的属性就是实体的属性，关系的码就是实体的码。
  （1）一个1:1联系可以转换为一个独立的关系模式，也可以与任意一端对应的关系模式合并（系表、系主任表。一个系只有一个系主任，他们之间的联系）
  （2）一个1:n联系可以转换为一个独立的关系模式，也可以与n端对应的关系模式合并。一般采用后者，前者多冗余。
  （3）一个m:n联系转换为一个关系模式。与该联系相连的各实体的码以及联系本身的属性均转换为关系的属性，各实体的码组成关系的码或关系码的一部分（Sno,Cno）
  （4）三个或三个以上实体间的一个多元联系可以转换为一个关系模式。与该多元联系相连的各实体的码以及联系本身的属性均转换为关系的属性，各实体的码组成关系的码或关系码的一部分
  （5）具有相同码的关系模式可合并

  2、数据模型的优化（规范化）
  3、设计用户子模式

关系查询处理和查询优化

- 关系数据库管理系统查询处理
  查询分析：对查询语句进行扫描、词法分析（区分select与slect）、语法分析（是不是select where from的结构），进行语法检查和语法分析，即判断查询语句是否符合语法规则。
  查询检查：对合法的查询语句进行语义检查，即根据数据字典中有关的模式定义检查语句中的数据库对象（关系名、属性）是否存在；若是对视图的操作则要执行视图消解方法把对视图的操作转换为对基本表的操作；还要检查数据字典中的用户权限定义对用户的存取权限进行检查。检查过后便把sql查询语句转换为内部表示，即等价的关系代数表达式。
  查询优化：代数优化（关系代数表达式的优化，按照一定的规则通过对关系代数表达式进行等价变换，改变代数表达式中操作的次序和组合，使查询执行更高效）和物理优化（指存取路径和底层操作算法的选择）
  查询执行：根据优化器得到的执行策略生成查询执行计划，由代码生成器生成执行这个查询计划的代码，然后加以执行，回送查询结果。
- 查询优化
  - 优点在于用户不必考虑如何最好地表达查询以获得较高的效率，而且在于系统可以比用户程序的优化做得更好。因为：
    1、优化器可以从数据字典中获得许多统计信息（例如关系表中的元组数、属性值的分布情况、索引建立情况）。而用户程序难以获取。
    2、如果数据库的物理统计信息改变了，系统可以自动对查询进行重新优化以选择相适应的执行计划。而在非关系系统中则必须重写程序，而重写程序在实际应用中往往不太可能。
    3、优化器可以考虑几百种不同的执行计划，程序员只能考虑几种有限的可能性。
    4、优化器中包括了很多复杂的优化技术，而这些优化技术往往只有最好的程序员才能掌握。
  - 代数优化
    - 查询树的启发式优化：
      1、选择运算尽可能优先做（减少元组数量）
      2、把投影和选择运算同时进行，因为没必要重复扫描
      3、把投影和其前后的双目运算结合起来，没有必要为去掉某些字段而扫描一遍关系
      4、把某些选择同在他前面要执行的笛卡尔积结合起来成为一个连接运算
      5、找出公共子表达式。
  - 物理优化

数据库恢复技术

- 事务
  - 基本概念：事务是用户定义的一个操作序列，这些操作要么全做，要么全不做
  - 事务是恢复和并发控制的基本单位
  - ACID特性：
    1、原子性：要么都做，要么都不做
    2、一致性：事务执行的结果必须是使数据库从一个一致性状态变到另一个一致性状态
    3、隔离性：一个事务的执行不能被其他事务打扰，并发执行的各个事务之间也不能相互打扰
    4、持续性：一个事务一旦提交，它对数据库中数据的改变就应该是永久性的。接下来的其他操作或故障不应该对其执行结果有任何影响。
- 故障种类
  - 事务内部的故障：造成不一致性了，回滚撤销。（转账，我的钱少了，你的钱没多）
  - 系统故障
    - 还未完成的事务可能有部分已经写到外存了，回滚
    - 已经完成的事务修改的结果可能还在缓冲区，重做
- 恢复的实现技术
  - 数据转储技术
    - 海量转储与增量转储
    - 静态转储与动态转储
  - 登记日志文件
- 恢复策略
  - 事务内部的故障：对照日志文件从后往前扫描，依次做逆操作。直至找到开始标志。
  - 系统故障：对照日志文件从前往后扫描，要重做的加入重做队列，要回滚的加入回滚队列
- 具有检查点的恢复技术
- 数据库镜像（特点：实时）
- 思考：
  1、试述事务的概念及事务的4个特性。恢复技术能保证事务的哪些特性。
  2、为什么事务非正常结束会影响数据库数据的正确性？请举例说明之。
  3、登记日志文件为什么必须先写日志文件，后写数据库
  4、针对不同的故障，试给出恢复的策略和方法
  5、什么是检查点记录？检查点记录包括哪些内容？
  6、具有检查点的恢复技术有什么优点？试举一个具体例子加以说明。
  7、试述使用检查点方法进行恢复的步骤
  8、什么是数据库镜像？他有什么用途？

并发控制技术

- 事务是并发控制的基本单位。为了保证事务的隔离性和一致性，数据库管理系统需要对并发操作进行正确调度。
- 不一致性问题
  - 丢失修改：两个事务读入同一数据并修改。一个事务修改的结果，被另一个事务提交的结果破坏。如买机票。
  - 不可重复读：事务T1读数据后，事务T2执行数据更新操作，数据或修改、或丢失、或新增，T1再次读这个数据时无法再现前一次读的结果。
  - 读脏数据：T1修改某一数据并将其写会磁盘，T2读数据后，T1因某种原因被撤销，被T1修改过的数据恢复原样，这时T2读到的数据与数据库不一致，这就是脏数据
- 封锁
  - 排它锁、写锁、X锁：事务T给数据对象A加上X锁，则仅事务T可以读取和修改A，其他事务仅可读取数据对象A，但不能读取和修改数据对象A。且其他事务此时不能再对数据对象A加锁。
  - 共享锁、读锁、S锁：事务T给数据对象A加上S锁，则事务T可以读取但不能修改A，其他事务只能对A加S锁不能加X锁。
- 封锁协议
  - 一级封锁协议：事务T在修改数据R之前必须对R加上X锁，直到事务结束后才释放。
    上锁期间，其它事务不能对数据R修改，解决了“丢失修改问题”，但其他事务仍可以读，没有解决“脏读”问题以及“不可重复读”问题
  - 二级封锁协议：在一级封锁协议的基础之上，事务T在读取数据R之前必须对R加上S锁，直到读取结束后才释放。
    由于事务T在读取数据之前必须对R加上S锁，其他事务在修改R期间有X锁的存在，事务T无法对R加S锁，也就无法读取R，解决了“脏读”问题。又由于在读完就释放锁，因此没有解决”不可重复读“问题
  - 三级封锁协议：在一级封锁协议的基础之上，事务T在读取数据R之前必须对R加上S锁，直到事务结束后才释放。
    某事务执行期间，其他任何事务都改不了数据，解决了” 不可重复读 “问题
- 活锁和死锁
  - 活锁：事务T1封锁了数据R，事务T2也请求封锁数据R，于是T2等待；事务T3也请求封锁R，事务T1结束后，系统首先批准T3的请求.....T2永远等待，即活锁。
    避免活锁最简单的方法：先来先服务
  - 死锁：事务T1封锁了数据R1，事务T2封锁了数据R2.随后，T1请求R2，T2请求R1，造成一种相互等待的局面。
    - 死锁的预防
      - 一次封锁法：每个事务必须一次性把自己需要的数据加锁，否则就不能继续执行。但第一，加大了封锁范围，势必降低并发度；第二，数据库中数据是不断变化的，原本不要求封锁的对象很可能在执行过程中会变成封锁对象，为此只能加大封锁范围，将事务执行过程中可能需要封锁的对象也封锁，这就进一步降低了并发度。
      - 顺序封锁法：预先对数据对象规定一个封锁顺序，所有事务都按这个顺序实施封锁。这可以有效防止死锁，但也有问题。第一，数据库中需要封锁的对象极多且可能是动态变化的，维护这样的封锁顺序开销很大；第二，事物的封锁请求可以随着事务的动态执行而动态决定，很难事先确定每一个事务要封锁哪些对象，也就很难按规定的顺序请求封锁资源。
    - 死锁的诊断和解除
      - 超时法：如果一个事务的等待时间超出了规定的时限，则认为其处于死锁状态。但，这个规定时限太长会导致不能及时发现死锁，太短会导致误判。
      - 等待图法：每个结点表示运行的事务，边表示事务等待的情况。若存在回路，则表示系统出现了死锁。
- 并发调度的可串行性
  - 冲突操作：不同的事务对同一个数据的读写操作和写写操作（不同事务的冲突操作和同一事务的两个操作是不能交换的，如某一事务对x的读和写。）
  - 可串行化调度：多个事务的并发执行是正确的，当且仅当其结果与按某一次序串行地执行这些事务时的结果相同，则为可串行化调度。
  - 冲突可串行化调度：通过交换两个事务不冲突操作的次序得到另一个调度，如该调度是串行的，则该调度是冲突可串行化调度。
  - 冲突可串行化调度一定是可串行化调度，但可串行化调度不一定是冲突可串行化调度。
- 两段锁协议：所有事务必须分两个阶段对数据项加锁和解锁
  - 为什么要引入-保证并发调度的正确性，采取手段保证调度是可串行化的
  - 内容：
    1、在对任何数据进行读、写操作之前，首先要申请并获得对该数据的封锁（但不能释放任何锁）
    2、在释放一个封锁之后，事务不再申请和获得任何其他封锁（但不能申请任何锁）
  - 若并发控制的事务都遵循两段锁协议，则对这些事务的任何调度策略都是可串行化的。但不是所有可串行化的策略都遵循两段锁协议。
- 封锁的粒度
  - 封锁对象的大小称为粒度。合适的粒度至关重要，它与系统的并发度和开销密切相关。
  - 多粒度封锁：首先定义多粒度树（数据库、关系、元组等）。检查效率低。
    - 显式封锁：应事务的要求直接加到数据对象上的锁
    - 隐式封锁：该数据对象没有被独立加锁，是由于其上级结点加锁而使该数据对象加上了锁
  - 意向锁：如果对一个节点加意向锁，这说明该节点的下层结点正在被加锁；对任一结点加锁时，其上层节点必须先加意向锁。
    IX、IS、SIX（自己要读表，但可能对某个元组修改）
- 思考：
  1、在数据库中为什么要并发控制？并发控制技术能保证事务的哪些特性？
  2、并发操作可能会产生哪几类数据不一致？用什么方法能避免各种不一致的情况？
  3、什么是封锁？基本的封锁类型有几种？试述它们的含义
  4、如何用封锁机制保证数据的一致性。
  5、什么是活锁？试述活锁的产生原因和解决方法
  6、什么是死锁？请给出预防死锁的若干方法
  7、请给出检测死锁的一种方法，当发生死锁后如何解除死锁
  8、什么样的并发调度是正确的调度？
  9、为什么若并发事务调度遵循两段锁协议则对这些事务的并发调度是可串行化的？
  10、举例说明对并发事务的一个调度是可串行化的，而这些事务不一定遵守两段锁协议。
  11、为什么要引入意向锁？意向锁的含义是什么？
  12、试述常用的意向锁：IS锁、IX锁、SIX锁，给出这些锁的相容矩阵