# 数据库系统

## 数据库模式

### 数据模式（Date Schema）

数据库存放数据的模式，正因为有数据模式，才能构造复杂的数据结构来建立数据之间的内在联系与复杂关系，从而构成数据的全局结构模式。

数据模式是基于选定的数据模型对数据进行“型”方面的刻画，而相应的“实例”则是对数据“值”方面的描述。先有数据模型，才能据其讨论相应的数据模式，有了数据模式，就能依据该模式得到相应的实例。

### 模式（Schema）

定义：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，是所有用户的公共数据视图。

理解：

1. 一个数据库只有一个模式
2. 是数据库数据在逻辑级上的视图
3. 数据库模式以某一种数据模型为基础
4. 定义模式时不仅要定义数据的逻辑结构（如数据记录由哪些数据项构成，数据项的名字、类型、取值范围等），而且要定义与数据有关的安全性、完整性要求，定义这些数据之间的联系。

### 外模式（External Schema）

定义：也称子模式（SubSchema）或用户模式，是数据库用户（包括应用程序员和最终用户）能够看见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。

理解：

1. 一个数据库可以有多个外模式
2. 外模式就是用户视图
3. 外模式是保证数据安全性的一个有力措施

### 内模式（Internal Schema）

定义：也称存储模式（Storage Schema），他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表现方式，（例如，记录的存储方式是顺序存储、按照B树结构存储还是按hash方法存储；索引按照什么方式组织；数据是否压缩存储，是否加密；数据的存储记录结构有何规定）。

理解：

1. 一个数据库只有一个内模式
2. 一个表可能由多个文件组成，如：数据文件、索引文件

他是数据库管理系统（DBMS）对数据库中数据进行有效组织和管理的方法

其目的有：

1. 为了减少数据冗余，实现数据共享
2. 为了提高存取效率，改善性能

### 概念模式（Conceptual Schema）

概念模式是数据库系统中全局数据逻辑结构的描述，是全体用户（应用）公共数据视图，此种描述是一种抽象描述，他不涉及具体的硬件环境与平台，也与具体的软件环境无关。

概念模式主要是描述数据的概念记录类型及数据以及他们间的关系，他还包括一些数据间的语义约束，对他的描述可用DBMS中的DDL语言定义。

### 外模式解释

外模式也称子模式或称用户模式（User’s Schema）他是用户的数据视图，亦即是用户所见到的模式的一个部分，他由概念模式推导而出，概念模式给出了系统全局的数据描述而外模式则给出每个用户的局部描述。一个概念模式可以有若干个外模式，每个用户只关心与他有关的模式，这样可以屏蔽大量无关信息且有利于数据的保护，因此对用户极为有利。在一般的DBMS中都提供有相关的外模式描述语言（外模式DDL）。

### 内模式解释

内模式又称为物理模式（Physical Schema），他给出了数据库物理存储结构与物理存取方法，如数据存储的文件结构、索引、集簇及hash等存取方式与存取路径，内模式的物理性主要体现在操作系统及文件级上，他还深入到设备级上，（如：磁盘及磁盘操作），但近年来有向设备级发展的趋势（如原始硬盘、磁盘分块技术等），DBMS一般提供相关的内模式描述语言（内模式DDL）。

数据模式给出了数据库的数据框架结构，而数据库中的数据才是真正的实体，但这些数据必须按框架所描述的结构组织，以概念模式为框架所组成的数据库叫概念数据库（Conceptual Database），以外模式为框架所组成的数据库叫用户数据库（user’s Database），以内模式为框架所组成的数据库叫物理数据库（Physical Database），这三种数据库中只有物理数据库是真实存在与计算机外存中，其他两中数据库并不真正存在与计算机中，而是通过两种映射由物理数据库映射而成。

模式的三个级别层次反映了模式的三个不同环境以及他们的不同要求，其中内模式处于最低层，他反映了数据在计算机物理结构中的实际存储形式，概念模式处于中层，他反映了设计者的数据全局逻辑要求，而外模式处于最外层，他反映了用户对数据的要求。

数据库系统的三级模式是对数据的三个级别的抽象，他把数据的具体物理实现留给物理模式，使用户与全局设计者能不必关系数据库的具体实现与物理背景，同时，他通过两级映射建立三级模式间的联系与转换，使得概念模式与外模式虽然并不具物理存在，但是也能通过映射而获得其存在的实体，同时两级映射也保证了数据库系统中数据的独立性，亦即数据的物理组织改变与逻辑概念改变，并不影响用户外模式的改变，他只要调整映射方式而不必改变用户模式。

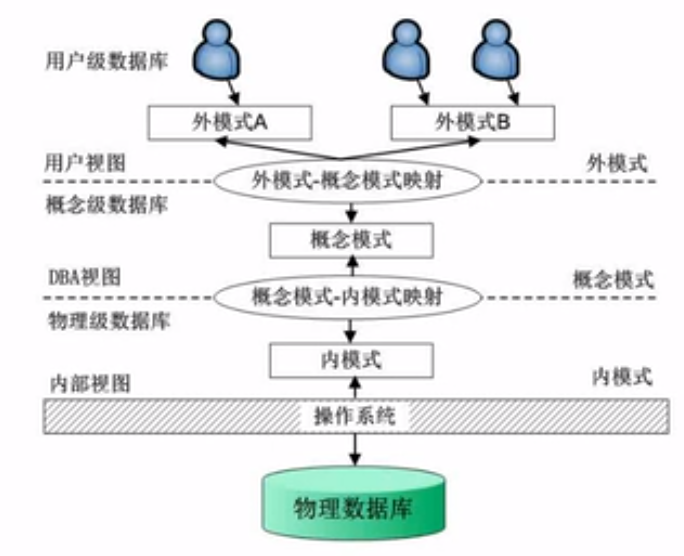
### 概念模式到内模式的映射

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，此种映射一般由DBMS实现。

### 外模式到概念模式的映射

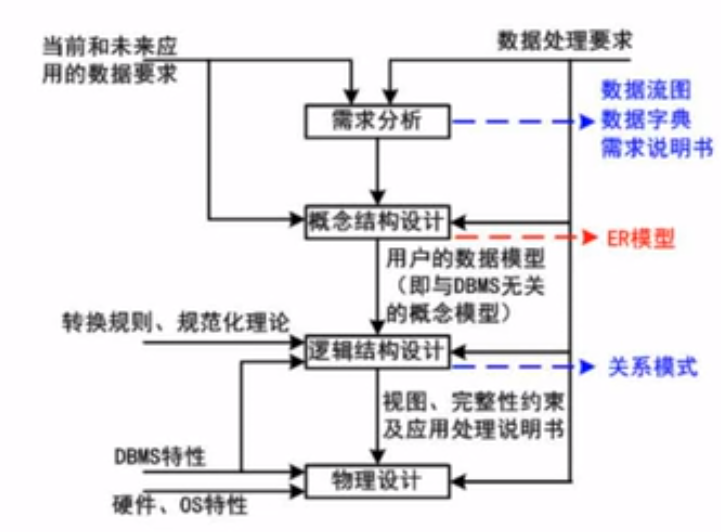
概念模式是一个全局模式而外模式则是用户的局部模式，一个概念模式中可以定义多个外模式，而每个外模式是概念模式的一个基本视图，外模式到概念模式的映射给出了外模式与概念模式的对应关系，这种映射一般由DBMS实现。

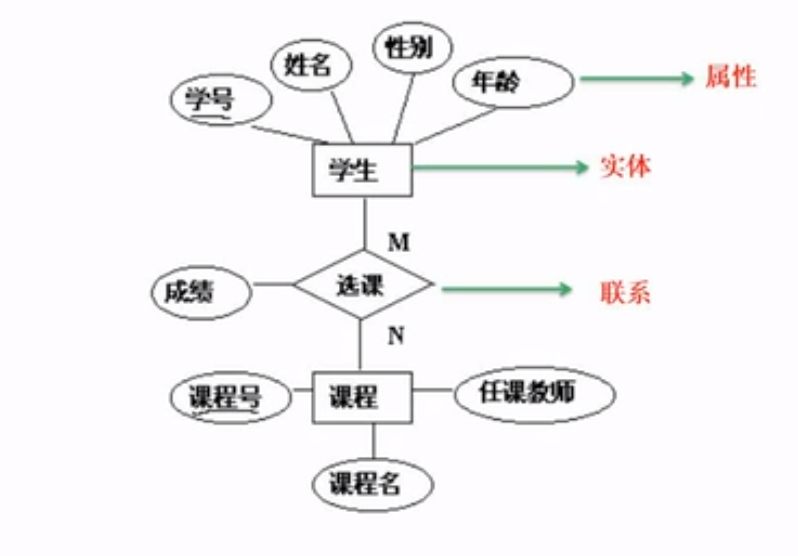
### 三级模式-两级映射



## ER模型

### 数据库设计过程





### 集成的方法：

多个局部E-R图一次集成

逐步集成，用累加的方式一次集成两个局部E-R

集成产生的冲突及解决办法：

属性冲突：包括属性域冲突和属性取值冲突

命名冲突：包括同名异义和异名同义

结构冲突：包括同一对象在不同应用中具有不同的抽象，以及同一实体在不同局部E-R图中所包含的属性个数和属性排列次序不完全相同。

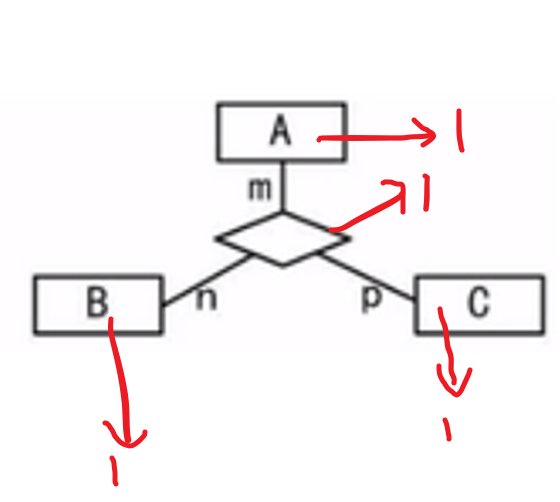
一个实体型转换为一个关系模式

1：1联系 1：n联系 m：n联系

三个以上实体间的一个多元联系

### 例题

在数据库逻辑结构的设计中，将E-R模型转换为关系模型应遵循相关原则。对于三个不同实体集和它们之间的多对多联系m：n：p，最少可转换为（4）个关系模式。



### 概念

我们目前所接触的数据库基本上是关系数据库，关系数据库中的关系模式是型，而关系是值。关系模式是对关系的描述。

什么是关系？关系实质上是一张二维表，其中每一行是一个元组，每一列是一个属性，每个元组是该关系涉及到属性集合笛卡尔积的一个元素。（笛卡尔积是这样的一个集合，集合中的元素是有序对，若A=｛0，1｝，B=｛a，b｝则A\*B = ｛<0,a>,<0,b>,<1,a>,<1,b>｝）关系是元组的集合，所以关系模式要描述元组的集合，其中包括哪些属性，属性来自域，属性与域之间的映射关系。

关系模式可以用五元组形式表示R（U，D，Dom，F），其中R表示关系名，U表示属性集合，Dom表示属性域（来自哪个域），F：表示函数依赖。

但是一般情况下，我们通常把关系模式表示为：R（U）或者R（A，B）（其中A，B代表U中的属性）

那么E-R图转化成关系模式的步骤：

前提：是已经把需求中的实体，以及实体中的联系确定。

第一：把每个实体都转换成关系模式R（A，B）形式。（A，B代表属性）

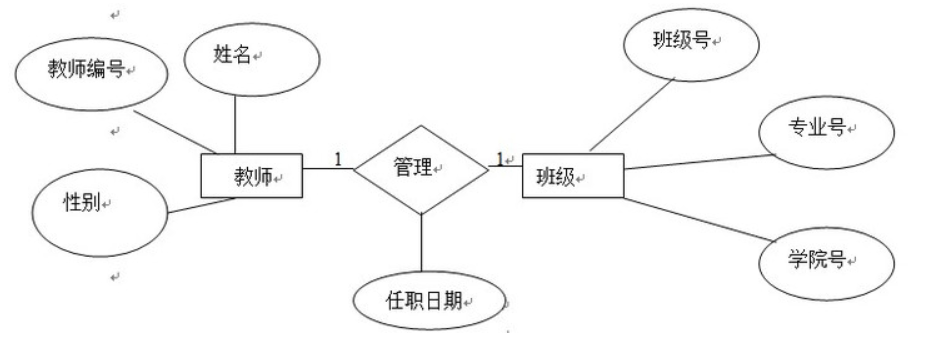
第二：实体中的属性即关系模式中的属性要求是满足第一范式（原子性，不可再分）

第三：也是最重要的步骤，实体之间联系的转换。

实体之间的联系分为：1：1，1：n，m：n三种形式。

#### 1:1之间的转换

将两个实体转换成关系模式之后，然后把任意实体的主键和联系的属性放到另一个实体的关系模式中。如图：



结果如下：

R1（教师编号、姓名、性别）

R2（班级号、专业号、学员号、教师编号、任职日期）

或者

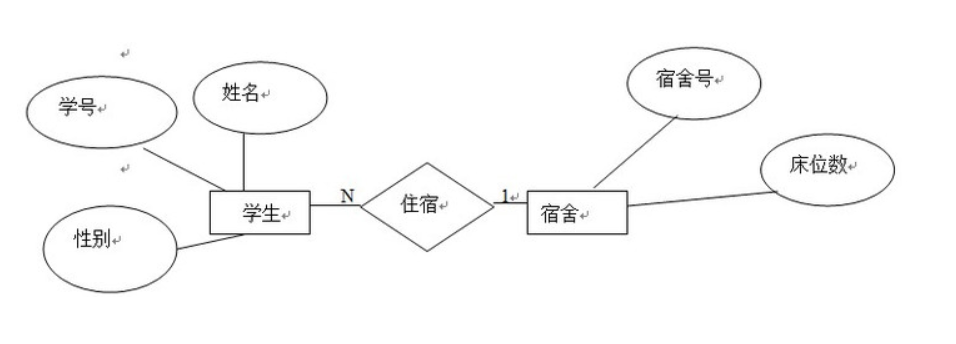
R1（教师编号、姓名、性别、班级号、任职日期）

R2（班级号、专业号、学员号）

#### 1:n之间的转换

将两个实体各自转换成关系模式后，把联系数量为1的实体的主键和联系的属性放到联系数量为n的实体关系模式中。

比如：学生和宿舍之间的关系。



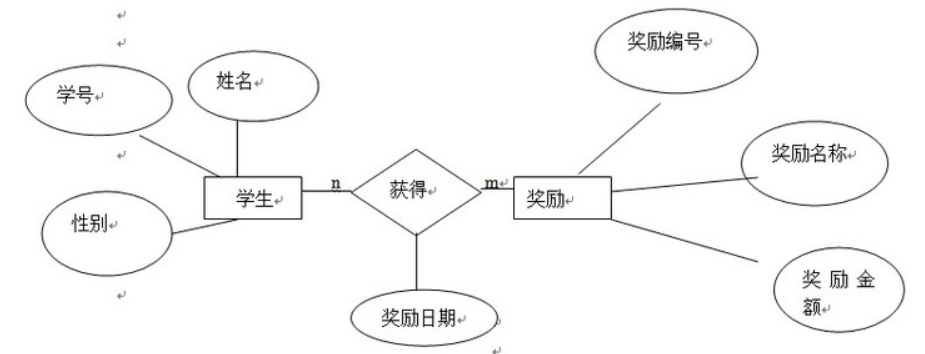
结果如下：

R1（学号、姓名、性别、宿舍号）

R2（宿舍号、床位数）

#### m:n的转换

将两个实体各自转换成关系模式后，把两个实体中的主键和联系的属性放到另一个（新的）关系模式中。（注意多生成一个关系模式）



结果如下：

R1（学号、姓名、性别）

R2（奖励编号、奖励名称、奖励金额）

R3（学号、奖励编号、奖励日期）

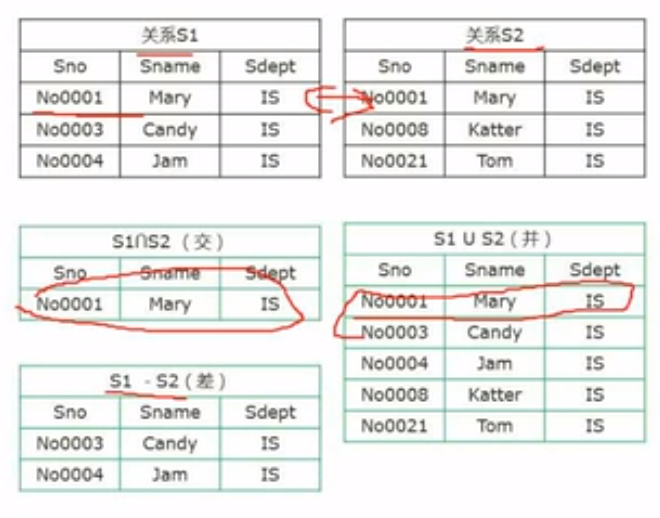
注：E-R图中联系没有联系属性，EE-R图中含有联系属性。

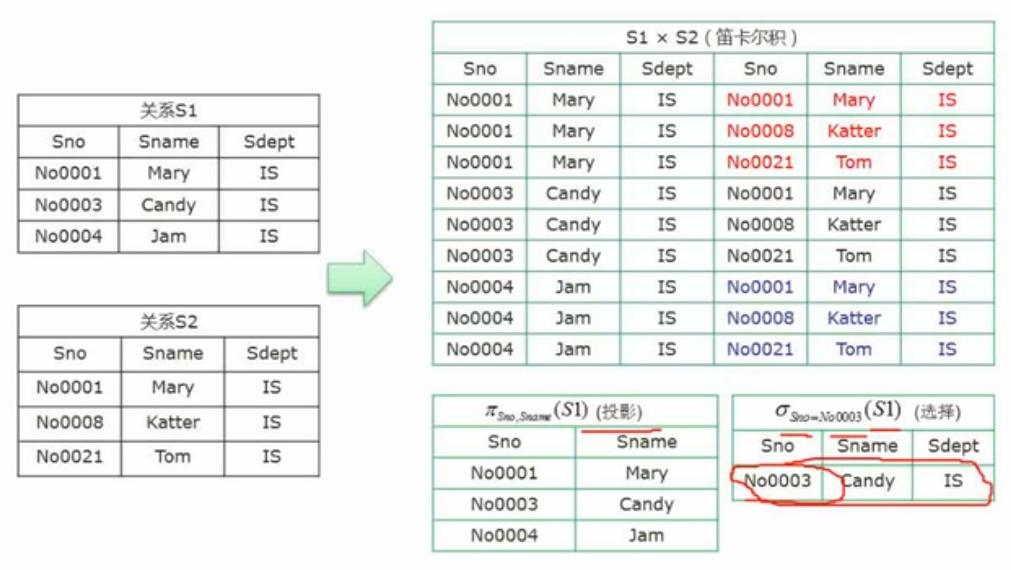
含有下划线的属性代表主属性，在表中当主键，红字体代表外键。

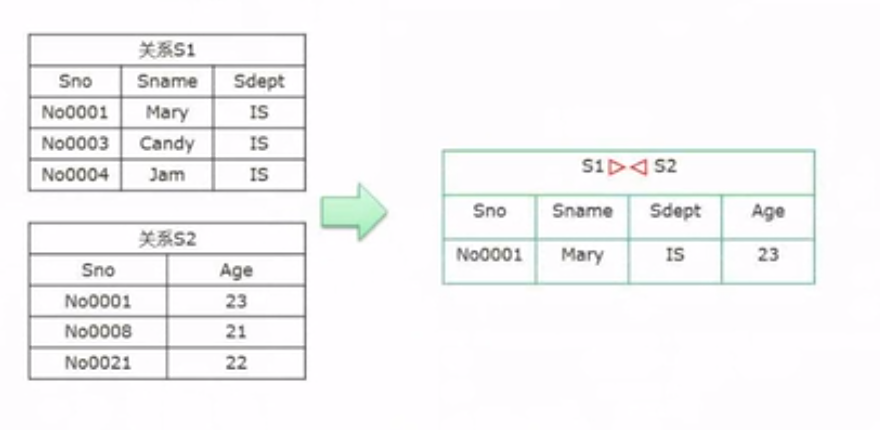
## 关系代数与元祖演算

### 关系代数

并、交、差、笛卡尔积、投影、选择、联接







## 规范化理论

一个关系数据库由一组关系模式组成，一个关系由一组属性名组成，关系数据库设计就是如何把已给定的相互关联的一组属性名分组，并把每一组属性名组织成关系的问题。

### 1、关系规范化的作用

所谓规范化，就是用形式更为简洁、结构更加规范的的关系模式取代原有关系的过程。

### 2、函数依赖

#### 2.1、属性间的关系

实体间的联系有两类：一类是实体与实体之间的联系，另一类是实体内部各属性间的联系

属性间的联系可分为以下三类：

1. 一对一关系（1：1）

以职工模式为例：职工（职工号、姓名、职称、部门）。如果该企业（或单位）中职工无重名，则属性职工号和姓名之间是1：1联系。一个职工号唯一的决定一个姓名，一个姓名也可以决定唯一的职工号。

设x，y是关系R的两个属性（集）。如果对于x中的任意具体指，y中至多有一个值与之对应，且反之亦然，则称x、y两属性间是一对一关系。

1. 一对多联系（1：m）

在职工模式中，职工号和职称间是一对多联系。一个职工号值对应一种职称（如张三只能对应工程师），但一种职称却可对应多个职工号（如工程师可对应多名职工）。

设x，y是关系R的两个属性（集）。如果对于x中的任一具体指，y中至多有一个值与之对应，而y中的一个值却可以和x中的n个值相对应，则称y对x是一对多联系。

1. 多对多联系（m:n）

在职工模式中，职称和部门之间是多对多联系，一种职称可分布在多个部门中（如每一个部门均有工程师），而一个部门中也可有多个职称。

设x，y是关系R的两个属性（集）。如果对于x中的任一具体指，y中有m个值与之对应，而y中的一个值也可以和x中的n个值相对应，则称y对x是多对多联系。

上述属性间的三种联系实际上是属性值之间相互依赖又相互制约的反映，称为属性间的数据依赖。

数据依赖共有三种，函数依赖（FuntionalDependency，简称FD）、多值依赖（Multivalued Dependency，简称MVD）和连接依赖（JoinDependency，简称JD），其中最重要的是函数依赖和多值依赖。

#### 2.2、函数依赖

函数依赖是属性间的一种联系。假定给定一个属性值，就可以唯一确定（查到）另一个属性的值。

定义：所谓函数依赖是指在关系R中，x，y为R的两个属性或属性组，如果对于R的任一关系r都存在：对于x的每一个具体指，y都只有一个具体指与之对应，则称属性y函数依赖于x。或者说属性x函数决定属性y，记做XàY。其中x叫决定因素，y叫被决定因素。当y是x的子集时，称为平凡函数依赖。由于平凡函数依赖总是成立的，因此，若不作特殊声明，本书后面提到的函数依赖，都不包含平凡函数依赖。

此定义可简单表述为：如果属性x的值决定属性y的值，那么属性y函数依赖于属性x。

前面讨论的属性间的三种联系，并不是每一种联系都存在函数依赖。

1. 如果两属性集x、y之间是1：1联系则存在函数依赖Xß>Y
2. 如果两属性集x、y之间是m：1联系则存在函数依赖XàY
3. 如果两属性集x、y之间是m：n联系则不存在函数依赖

#### 2.3、码的定义

定义：设K是关系模式R（U、F）中的属性或属性组，K‘是K的任一真子集。若KàU，而不存在K′àU，则K为R的候选码（CandidateKey），简称为码。

若候选码多于一个，则选定其中一个为主码（PrimaryKey）；

包含在任一候选码中的属性，叫做主属性（PrimaryAttribute）；

不包含在任何候选码中的属性称为非主属性，（NonprimeAttribute）或或非码属性（Non KeyAttribute）；

关系模式中，最简单的情况，单个属性是码，称为单码（SingleKey）；最极端的情况，整个属性组是码，称为全码（AllKey）。

定义：设有两个关系模式R和S，x是R的属性或属性组，但x是S的码（或与S的码意义相同），则称x是R的外部码（ForeignKey），简称外码

#### 2.4、函数依赖和码的唯一性

码是由一个或多个属性组成的可唯一标识元祖的最小属性组。码在关系中总是唯一的，即码函数决定关系中的其他属性。因此，一个关系中，码值总是唯一的（如果码的值重复，则整个元祖都会重复）。否则违反实体完整性规则。

与码的唯一性不同，在关系中，一个函数依赖的决定因素可能是唯一的，也可能不是唯一的。如果我们知道A决定B，且A和B在同一关系中，但我们仍然无法知道A是否能决定处B以外的其他所有属性，所以无法知道A在关系中是否是唯一的。

### 3、关系模式的规范化

#### 3.1、关系模式的规范化

当一个关系中的所有分量都是不可分的数据项时，该关系是规范化的。

关系按其规范化程度从低到高可分为5级范式，分别称为1NF（FirstNormalForm）、2NF、3NF（BCNF）、4NF、5NF。规范化程度较高必是较低者的子集。

#### 3.2、第一范式（1NF）

定义：如果关系模式中R不包含多值属性，则R满足第一范式，简称1NF（FirstNormalForm），记做R属于1NF。

1NF是规范化的最低要求，不满足1NF的关系是非规范化关系。

#### 3.3、第二范式（2NF）

定义：设X、Y是关系R的两个不同属性或属性组，且X属于Y。如果存在X的某一个真子集X‘，是X’属于Y成立，则称Y部分函数依赖与X，反之则称Y完全函数依赖于X。

定义：如果一个关系R属于1NF，且它的所有非主属性都完全依赖于R的任一候选码，则R属于第二范式，记做R属于2NF。

推论：如果关系模式R-1NF，且他的每一个候选码都是单码，则R属于2NF。

#### 3.4、第三范式（3NF）

定义：在关系R中，X、Y、Z是R的三个不同的属性或属性组，如果XàY，YàZ，但Y/-->X且Y不是X的子集，则称Z传递依赖于X。

定义：如果关系模式R属于2NF，且他的每一个非主属性都不传递依赖于任何候选码，则称R是第三范式，记做R属于3NF。

推论1：如果关系模式R属于1NF，且他的每一个非主属性既不部分依赖，也不传递依赖于任何候选码，则R属于3NF。

推论2：不存在非主属性的关系模式一定为3NF。

#### 3.5、改进的3NF——BCNF

定义：设关系模式R（U、F）属于NF，若F的任一函数依赖XàY（Y不是X的子集）中X都包含了R的一个码，则称R属于BCNF。换言之，在关系模式R中，如果每一个决定因素都包含码，则R属于BCNF。

由BCNF的定义可以得到以下推论：如果R属于BCNF，则：

R中所有非主属性对每一个码都是完全函数依赖；

R中所有所有主属性对每一个不包含他们的码都是完全函数依赖；

R中没有任何属性完全函数依赖于非码的任何一组属性。

定理：如果R属于BCNF则R属于3NF一定成立。

一个关系模式如果达到了BCNF，那么在函数依赖范围内，他已经实现了彻底的分离，消除了数据冗余，插入和删除异常。

### 4、多值依赖和第四范式

定义：设R（U）是属性集U上的一个关系模式，X、Y、Z是U的子集，且Z=U-X-Y。如果R（U）的任一关系r，给定一对（x，z）值都有一组y值与之对应，这组y值仅仅决定于x值而与z值无关。则称y多值依赖与x，或x多值决定y，记做XààY。

定义中如果z为空集，则称XààY为平凡的多值依赖，否则为非平凡的依赖。

定义：如果关系模式R属于1NF，对于R的每个非平凡的多值依赖XààY（Y不是X的子集），x含有码，则称R是第四范式，即R属于4NF。

一个关系模式如果属于4NF，则一定属于BCNF，但一个BCNF的关系模式，不一定是4NF的，R中所有的非平凡多值依赖实际上是函数依赖。

### 5、关系的规范化度

关系规范化的目的是解决关系模式中存在的数据冗余、插入和删除异常、更新繁琐等问题。其基本思想是消除数据依赖中的不合适部分，使各关系模式达到某种程度的分离，使一个关系描述一个概念、一个实体或实体间的一种联系。因此，规范化的实质是概念的单一化。

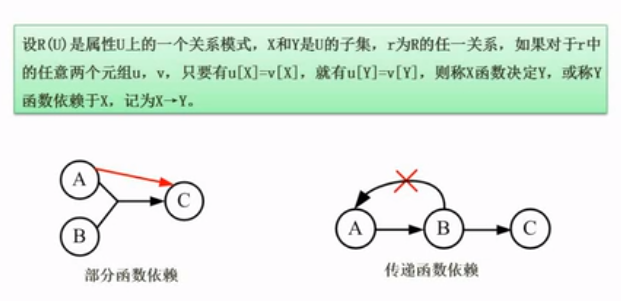
规范化的基本原则是：由低到高，逐步规范，权衡利弊，适可而止。通常，以满足第三范式为基本要求。

把一个非规范化的数据结构转换称第三范式，一般经过以下几步：

1. 把该结构分解成若干个属性第一范式的关系；
2. 对那些存在组合码，且有非主属性部分函数依赖的关系必须继续分解，使所得关系都属于第二范式
3. 若关系中有非主属性传递依赖于码，则继续分解之，使得关系都属于第三范式。

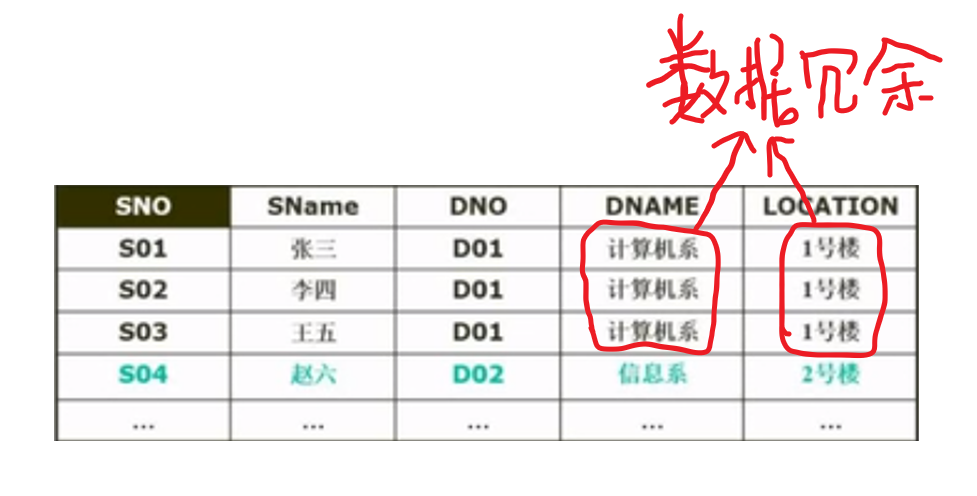
关系模式的规范化过程是通过投影分解实现的，即用投影运算把一个模式分解成若干个高一级的关系模式。这种投影分解不是唯一的。

### 6、函数依赖解析

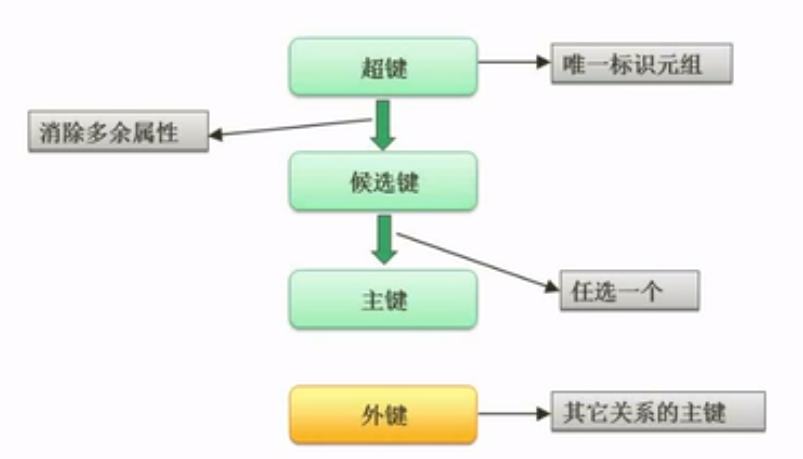


### 7、价值与用途

非规范化的关系模式，可能存在的问题包括：数据冗余、更新异常、插入异常、删除异常。



### 8、键

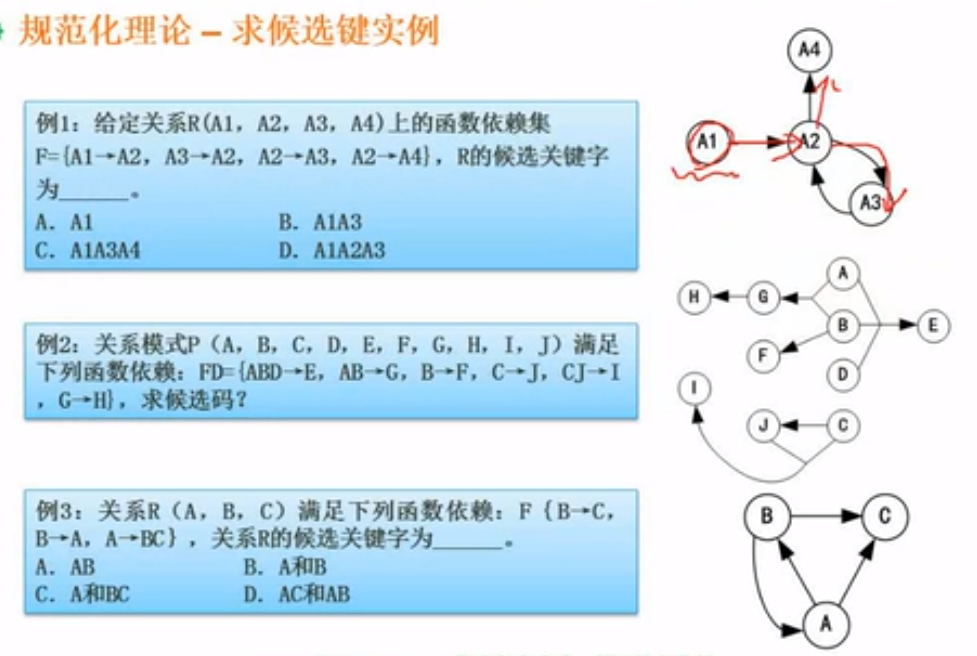


### 9、求候选键

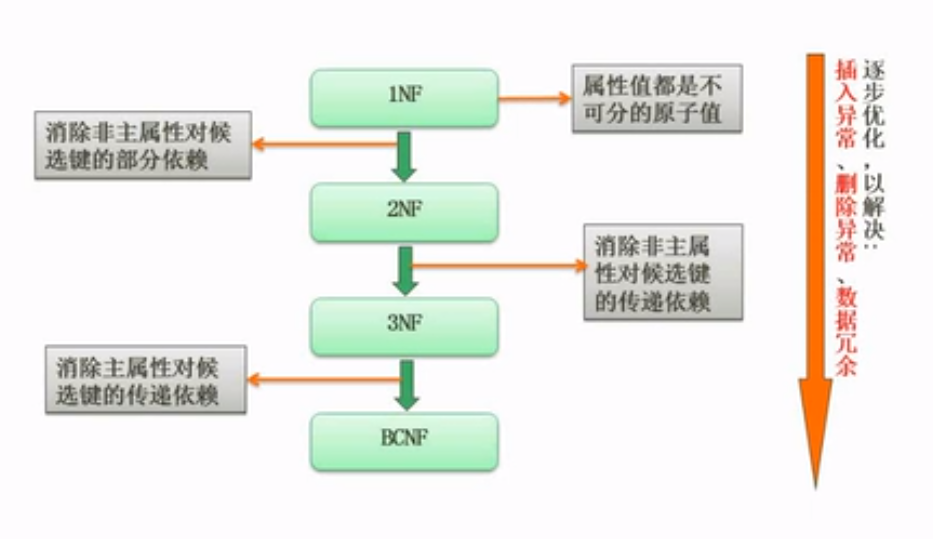
将关系模式的函数依赖关系用“有向图”的方式表示

找入度为0的属性，并以该属性集合为起点，尝试遍历有向图，若能正常遍历图中所有节点，则该属性集即为关系模式的候选键

若入度为0的属性不能遍历图中所有节点，则需要尝试性的将一些中间节点（既有入度，也有出度的节点）并入入度为0的属性集中，直至该集合能遍历所有结点，集合为候选键



### 10、范式



### 11、第一范式

第一范式（1NF）：在关系模式R中，当前仅当所有域只包含原子值，即每个分分量都是不可再分的数据项，则称R是第一范式。



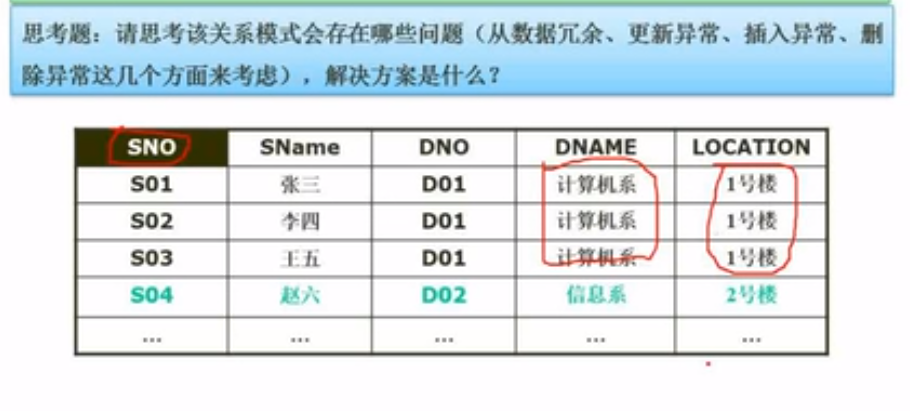
### 12、第二范式

第二范式（2NF）：当且仅当R是1NF，且每一个非主属性完全依赖主键（不存在部分依赖）时，则称R是第二范式。（2NF即在复合主键中，不存在主键一部分确定某一属性）



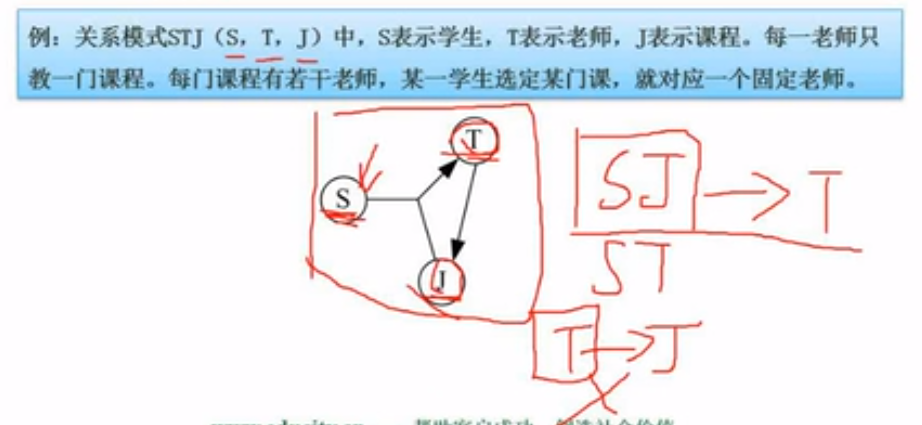
### 13、第三范式

第三范式（3NF）：当且仅当R是2NF，且E中没有非主属性传递依赖于码时，则称R是第三范式。（3NF即不存在非主属性确定非主属性）



### 14、BC范式

BC范式（BCNF）：设R是一个关系模式，F是他的依赖集，R属于BCNF当且仅当其F中每个依赖的决定因素必定包含R的某个候选码。



### 15、例题



### 16、模式分解

#### 保持函数依赖分解

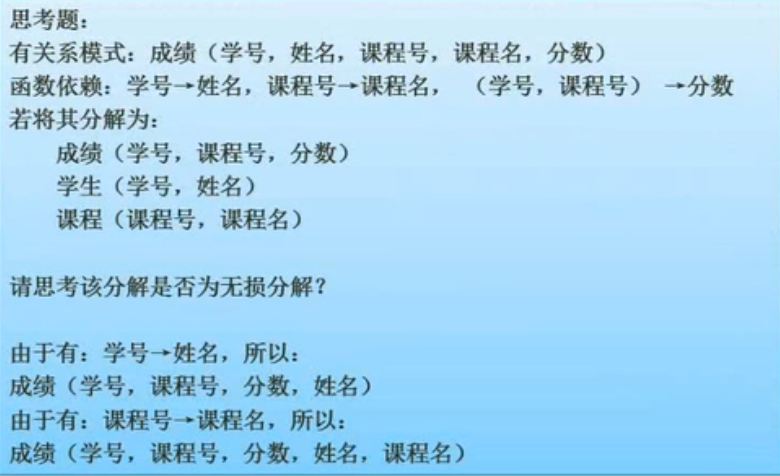
设数据库模式是关系模式R的一个分解，F是R上的函数依赖集，中每个模式Ri上的FD集是Fi。如果（F1，F2，…，Fk）与F是等价的（即相互逻辑蕴含），那么称分解保持FD。

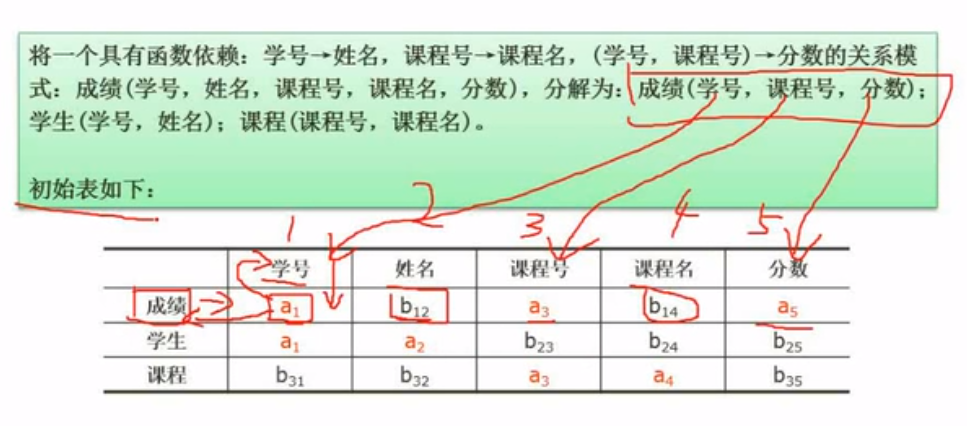
#### 无损分解

什么是有损，什么又是无损。

有损：不能还原，无损：可以还原。

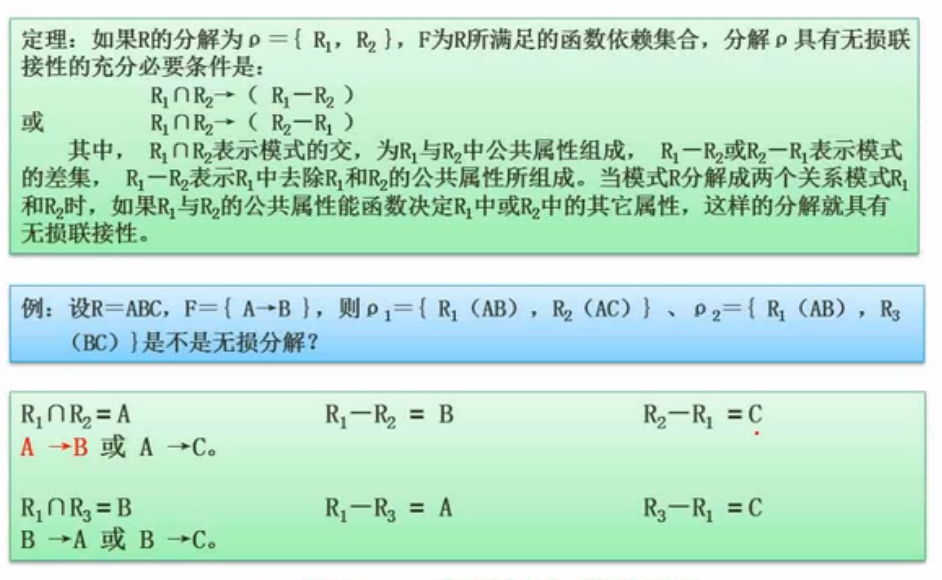
无损联接分解：指将一个关系模式分解成若干个关系模式后，通过自然联接和投影等运算还能还原到原来的关系模式。



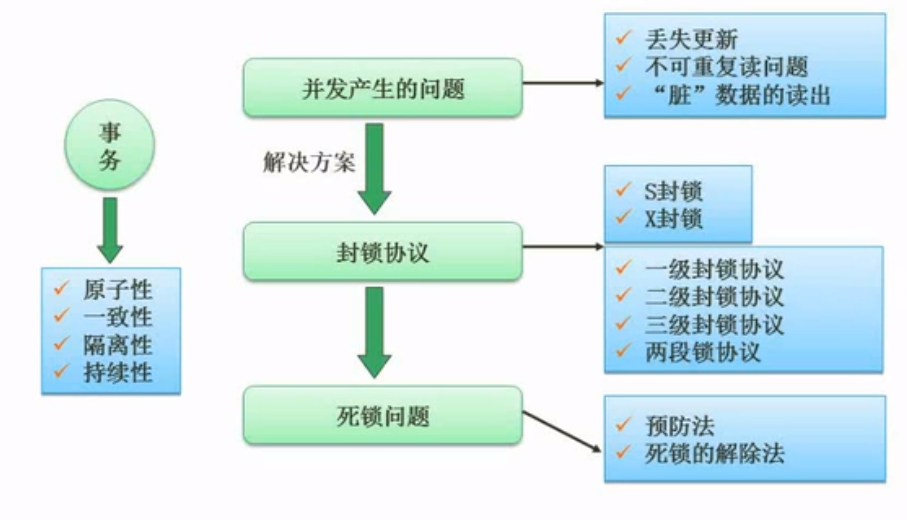


a 表示关系拥有该属性，b表示关系没拥有该属性

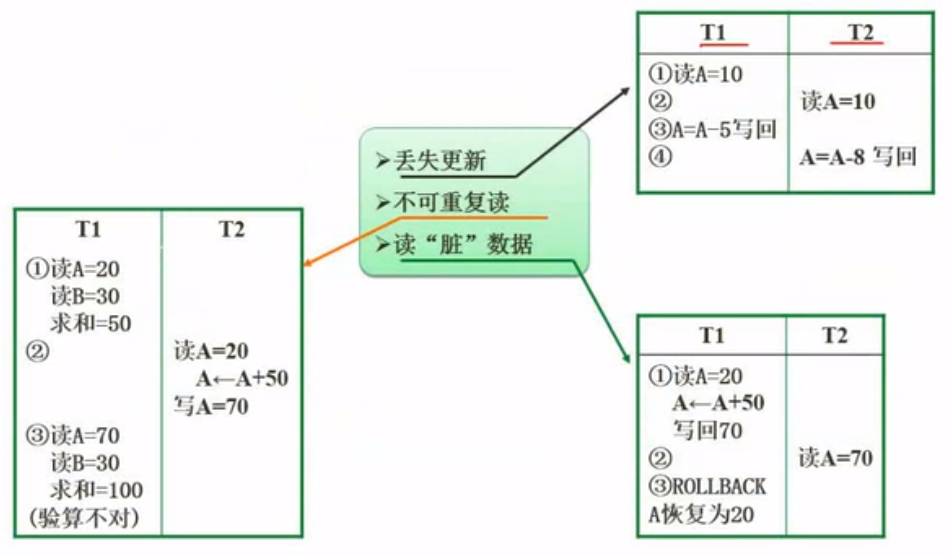




## 并发控制



### 存在问题示例



### 封锁协议

一级封锁协议：事务T在修改数据R之前必须先对其加X锁（写锁），直到事务结束才释放。可防止丢失修改。

二级封锁协议：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（读锁），读完后即可释放S锁。可防止丢失修改，还可防止读“脏”数据

三级封锁协议：一级封锁协议加上事务T在读取数据R之前对其加S锁，直到事务结束才释放。可防止丢失修改、防止读“脏”数据与防止数据重复读

两段锁协议：可串行化的。可能发生死锁。

### 脏读（dirty read）

A事务读取了B事务尚未提交的更改数据，并且在这个数据的基础上进行操作。如果此时恰巧B事务进行回滚，那么A事务读到的数据根本是不被承认的。

以下是一个取款事务和转账事务并发时引起的脏读场景。

|  |  |  |
| --- | --- | --- |
| 时间 | 转账事务A | 取款事务B |
| T1 |  | 开始事务 |
| T2 | 开始事务 |  |
| T3 |  | 查询账户余额为1000 |
| T4 |  | 取出500元，把余额改为500 |
| T5 | 查询账户余额为500元（脏读） |  |
| T6 |  | 撤销事务，余额恢复为1000元 |
| T7 | 汇入100元，余额改为600元 |  |
| T8 | 提交事务 |  |

在这个场景中，B希望取款500元，而后撤销了动作，而A往同一个账户转账100元，因为A事务读取了B事务尚未提交的数据，因而导致账户白白丢失了500元。在oracle中，不会发生脏读的情况。

### 不可重复读（unrepeatable read）

不可重复读是指A事务读取了B事务已经提交的更改数据。假设A在取款事务的过程中，B往该账户转账100元，A两次读取账户的余额发生不一致。

|  |  |  |
| --- | --- | --- |
| 时间 | 取款事务A | 转账事务B |
| T1 |  | 开始事务 |
| T2 | 开始事务 |  |
| T3 |  | 查询账户余额为1000元 |
| T4 | 查询账户余额为1000元 |  |
| T5 |  | 取出100元，把余额改为900元 |
| T6 |  | 提交事务 |
| T7 | 查询账户余额为900元 |  |

在同一个事务中T4和T7时间点读取的账户余额不一致

### 幻象读（phantom read）

A事务读取B事务提交的新增数据，这时A事务将出现幻象读的问题。幻读一般发生在计算机统计数据的事务中。举个例子，假设银行系统在同一个事务中两次统计存款的总金额，在两次统计过程中，刚好新增了一个存款账户，并存入100元，这时两次的统计的总金额将不一致。

|  |  |  |
| --- | --- | --- |
| 时间 | 统计金额事务A | 转账事务B |
| T1 |  | 开始事务 |
| T2 | 开始事务 |  |
| T3 | 统计存款总额为10000元 |  |
| T4 |  | 新增一个存款账户，存款为100元 |
| T5 |  | 提交事务 |
| T6 | 再次统计存款总额为100100元（幻象读） |  |

如果新增的数据刚好满足事务的查询条件，那么这个新数据就会进入事务的视野，因而导致两次统计结果不一致的情况。

幻读和不可重复读时两个容易混淆的概念，前者是指读到了其他事务已经提交的新增数据，而后者是读到了已经提交事务的更改数据（更改或删除）。为了避免这两种情况，采取的策略是不同的：防止读到更改数据，只需对操作的数据添加行级锁，阻止操作过程中的数据发生变化，而防止读到新增数据，则往往需要添加一个表级锁-将整张表锁定，防止新增数据（Oracle使用多版本数据的方式实现）。

### 第一类丢失更新

A事务撤销时，把已经提交的B事务的更新数据覆盖了。这种错可能会造成很严重的问题。通过下面的账号取款转账就可以看出来。

|  |  |  |
| --- | --- | --- |
| 时间 | 取款事务A | 转账事务B |
| T1 | 开始事务 |  |
| T2 |  | 开始事务 |
| T3 | 查询账号余额为1000元 |  |
| T4 |  | 查询余额为1000元 |
| T5 |  | 汇入100元，把余额改为1100元 |
| T6 |  | 提交事务 |
| T7 | 取出100元，把余额改为900元 |  |
| T8 | 撤销事务 |  |
| T9 | 余额恢复为1000元（丢失更新） |  |

A事务在撤销时，“不小心”将事务B已转入账号的金额给抹去了。

### 第二类丢失更新

A事务覆盖B事务已经提交的数据，造成B事务所操作丢失

|  |  |  |
| --- | --- | --- |
| 时间 | 转账事务A | 取款事务B |
| T1 | 开始事务 |  |
| T2 |  | 开始事务 |
| T3 | 查询余额为1000元 |  |
| T4 |  | 查询余额为1000元 |
| T5 |  | 取出100元，把余额改为900元 |
| T6 |  | 提交事务 |
| T7 | 汇入100元，把余额改为1100元 |  |
| T8 | 提交事务 |  |
| T9 | 把余额改为1100元（丢失更新） |  |

在上面的例子，由于支票转账事务覆盖了取款事务对余额所做的更新，导致银行最后损失了100元，相反如果转账事务先提交，那么用户损失了100元。

### 数据库的锁

insert , update , delete , select for update 语句会隐式采用必要的锁定。

1. 行共享锁：select for update 语句隐式获取行共享锁
2. 行独占锁：insert , update , delete 语句隐式获取，或者通过lock table in row exclusive mode 获取行独占锁
3. 表共享锁：lock table in share mode 获取，防止其他独占锁获取，但是允许在表内拥有多个行共享锁和表共享锁
4. 表共享行独占锁：lock table in share row exclusive mode 获取
5. 表独占锁：lock table in exclusive mode

### Spring事务

不能被Spring AOP 事务增强的方法

|  |  |
| --- | --- |
| 动态代理策略 | 不能被事务增强的方法 |
| 基于接口的动态代理（JDK） | 除了public方法均不能，此外public static也不能增强 |
| CGLib动态代理 | private，static ，final方法 |

对于private方法，由于最终会被public方法封装后在开放给外部调用者，而public方法是可以事务增强的，所以基本没有什么问题。在实际开放中，最容易造成隐患的是基于cglib代理的public static 和 public final 方法。原因是他们本身是public的因此可以直接被外部调用，只要调用方没有上下文事务，这些方法就会以无事务的方式运行。

## 数据库完整性约束

实体完整性约束

参照完整性约束

用户自定义完整性约束

触发器

## 数据库安全

|  |  |
| --- | --- |
| 措施 | 说明 |
| 用户标识和鉴定（身份认证） | 最外层的安全保护措施，可以使用用户账户、口令及随机数检验等方式。 |
| 存取控制（权限控制） | 对用户进行授权，包括操作类型（如查找、插入、删除、修改等动作 ）和数据对象（主要是数据范围）的权限。 |
| 密码存储和传输 | 对远程终端信息用密码传输 |
| 视图的保护 | 对视图进行授权 |
| 审计 | 使用一个专用文件或数据库，自动将用户对数据库的所有操作记录下来。 |

## 数据备份

分类方式一：

冷备份也称为静态备份，是将数据库正常关闭，在停止状态下，将数据库的文件全部备份（复制）下来。

热备份也称为动态备份，是利用备份软件，在数据库正常运行的状态下，将数据库中的数据文件备份出来。

|  |  |  |
| --- | --- | --- |
| 备份方式\优缺点 | 优点 | 缺点 |
| 冷备份 | 非常快速的备份方式（只需复制文件）；容易归档（简单复制即可）；容易恢复到某个时间点上（只需将文件再复制回去）；能与归档方法相结合，做数据库“最佳状态”的恢复；低度维护，高度安全 | 单独使用时，只能提供到某一个时间点上的恢复；在实施备份的全过程中，数据库必须要作备份而不能做其他工作；若磁盘空间有限，只能复制到磁带等其他外部存储设备上，速度会很慢；不能按表或按用户恢复 |
| 热备份 | 可在表空间或数据库文件级备份，备份的时间短，备份时数据库仍可使用；可达到秒级恢复（恢复到某一时间点上）；可对几乎所有的数据库实体做恢复；恢复是快速的 | 不能出错，否者后果严重，若热备份不成功，所得结果不可用于时间点恢复；困难于维护，所以要特别小心，不允许“以失败告终” |

分类方式二：

完全备份：备份所有数据

差量备份：仅备份上一次完全备份之后变化的数据

增量备份 ：备份上一次备份之后变化的数据

1. 静态海量转储：在系统中无运行事务时进行，每次转储全部数据库
2. 静态增量转储：在系统中无运行事务时进行，每次只转储上一次转储后更新过的数据。
3. 动态海量转储：转储期间允许对数据库进行存取或修改，每次转储全部数据库
4. 动态增量转储：转储期间允许对数据库进行存取或修改，每次只转储上一次转储后更新过的数据。

日志文件：事务日志是针对数据库改变所做的记录，它可以记录针对数据库的任何操作，并将记录结果保存在独立文件中。

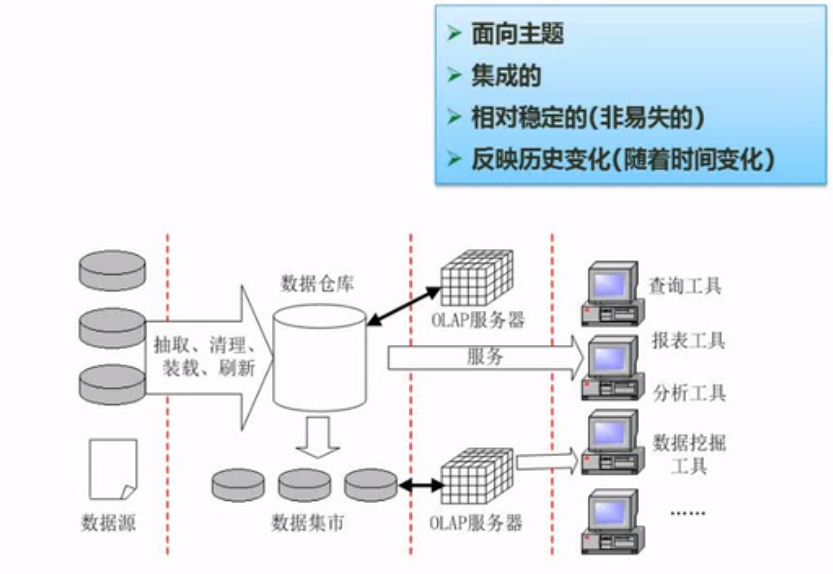


## 数据库故障与恢复

|  |  |  |
| --- | --- | --- |
| 故障关系 | 故障原因 | 解决方法 |
| 事务本身的可预期故障 | 本身逻辑 | 在程序中预先设置rollback语句 |
| 事务本身的不可预期故障 | 算术溢出、违反存储保护 | 由DBMS的恢复子系统通过日志，撤销事务对数据库的修改，回退到事务初始状态 |
| 系统故障 | 系统停止运转 | 通常使用检查点法 |
| 介质故障 | 外存被破坏 | 一般使用日志重做业务 |

## 分布式数据库

## 数据仓库与数据挖掘



### 数据挖掘方法分类

方法：

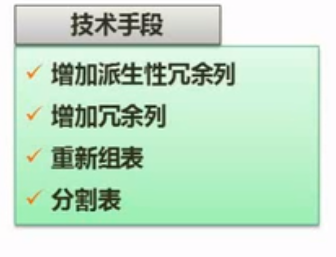
1. 决策树
2. 神经网络
3. 遗传算法
4. 关联规则算法

分类：

1. 关联分析：挖掘出隐藏在数据间的相互关系
2. 序列模式分析：侧重点是分析数据间的前后关系(因果关系)
3. 分类分析：为每一个记录赋予一个标记再按标记分类
4. 聚类分析：分类分析法的逆过程

## 反规范化

由于规范化会使表不断的拆分，从而导致数据表过多。这样虽然减少了数据冗余，提高了增、删、改的速度，但会增加查询的工作量。系统需要进行多次连接，才能进行查询操作，使得系统效率大大下降。



## 大数据

