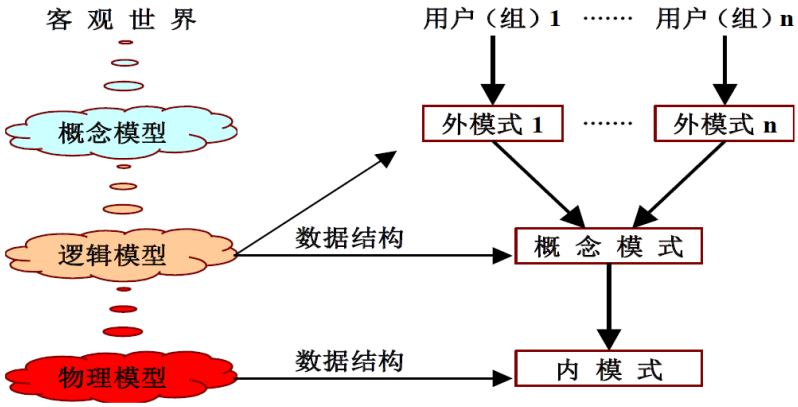
数据库概论

数据库系统概述

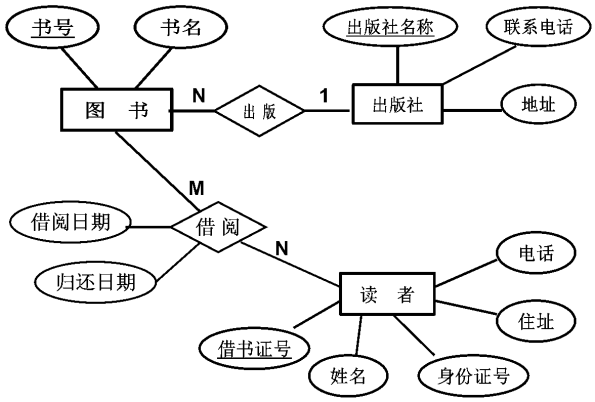
- 基本概念
  - 数据（D）
  - 数据库（DB）
  - 数据库管理系统（DBMS）
    - 数据定义语言（DDL）
    - 数据操纵语言（DML）
    - 数据控制语言（DCL）
  - 数据库管理员（DBA）
  - 数据库系统（DBS）
  - 数据库应用系统（DBAS）
- 数据库系统的特点
  - 数据的集成性
  - 数据的高共享性与低冗余性
  - 数据独立性
  - 数据统一管理与控制
- 数据库内部结构体系
  - 数据库三级模式
    - 外模式（用户数据库）->概念模式（概念数据库）->内模式（物理数据库）
  - 数据库二级映射
    - 从概念模式到内模式的映射：DBMS 实现
    - 从外模式到概念模式的映射：DBMS 实现

数据模型

- 数据模型的基本概念
  - 数据模型是数据基本特征的抽象，它描述①数据的结构；②定义在结构上的操作；③约束条件
  - 数据模型：概念模型、数据模型、物理模型
- 概念（数据）模型：与 DBMS、计算机系统平台均无关，侧重描述结构和关系
  - E-R 模型/实体-联系模型
  - EE-R 模型/拓展的实体-联系模型
  - 面向对象模型
  - 谓词模型
- 逻辑数据模型（DBMS 所提供的工具（DDL）来定义的数据模型）：概念模型转化成逻辑模型后在数据库中得以表示，面向数据库系统，着重于数据库系统一级实现
  - 层次模型、网状模型
  - 关系模型、面向对象模型、谓词模型
  - 对象-关系模型
- 物理（数据）模型：在计算机上的物理结构表示
- 三种数据模型和三级模式之间的关系



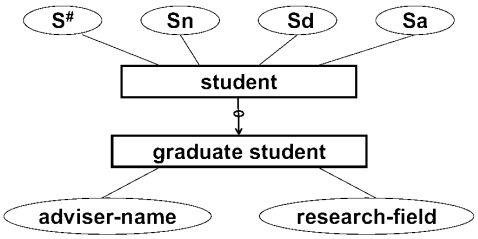
- 数据模型的 4 个世界
  - 现实世界：为整个转换过程提供客观基础与初始启动环境
  - 概念世界：与具体的 DBMS 和计算机无关
  - 信息世界：与具体的 DBMS 有关
  - 计算机世界：是 DB 的最终实现结构
- 概念世界与概念模型
  - 实体-联系模型/E(ntity)-R(elationship)模型：（要学会画图，期末考察）
    - 实体（概念世界基本单位）、属性（实体特征）、联系（实体集之间的关系）
    - 实体集：矩形；属性：椭圆；联系：菱形；画线：实线；函数对应关系：1:1、1:m（一对多）/m:1（多对一）、n:m 标到对应线段上
    - 注意要**对每一个主键添加下划线**！！！！！！
    - 若属性具有唯一性，则在对应属性下加一个下划线。例如：书号、出版社名称、借书证号具有唯一性，则需要加一个下划线



    - E-R 模型的设计选择
      - 实体？属性？
        - 实体：进一步多方面描述信息
        - 属性：单一描述值（非结构化的单值信息）
      - 实体？联系？
        - 实体：独立存在的持久对象
        - 联系：因为某种需要而产生，通常与多个对象有关
      - 二元联系？多元联系？
        - 如果①用户只需要使用两两联系，或者②不会出现歧义，那么可以考虑改用若干个二元联系实现
        - 否则基于涉及到的实体个数设计 n 元联系

- 属性依附对象：实体？联系?
  - 实体（集）的属性：内在属性，永久存在，不因联系的产生或消失而改变
  - 联系的属性：因联系产生而存在，随联系的消亡而消亡
  - 例如：学生的学号不因选课改变，但学生的成绩却因为选了课才有，不选课就没有，所以学号是实体的属性，成绩是联系的属性。
- 联系不能连接联系？是吗？，如果出现这种情况则考虑将其中一个改为实体集（如 exp_of_er_2.pdf 中）

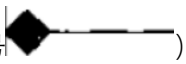
■ EE-R 模型：
- 增加了 Is-a 联系
  - 建立两个 A、B 两个实体集之间的继承关系，用父指向子的箭头中间加一个圈表示。
  - A Is-a B 中，A 是 B 的超（实体）集；B 是 A 的子（实体）集



- 增加了弱实体集
  - 实体 A 的存在需要依赖于其它实体集中某个实体的存在，多对一
  - 从弱实体到联系的有向箭头

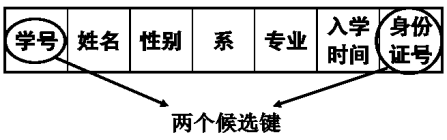
■ OO 模型（面向对象模型）
- Is-a 关系：子类继承超类；超类->子类为特化，子类->超类为普化
- Is-part-of 关系：复杂类分解为简单类，简单类聚合为复杂类
- 发送一条**消息**（跨对象的擦欧总），执行对象中的**方法**（接口和内部实现）
- UML 统一建模语言

（类的名字）
（属性） title　PK（表示主键） year　PK（表示主键） length genre
（方法）

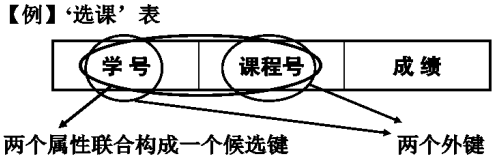
- 关联 m..n 表明一端至少有 m 个对象，至多有 n 个对象与另外一段端的对象连接；*代表无限；单独的*代表 0..*
- 子类：符号： 
- 聚集（aggregation）：空菱形那一端表示 0..1（符号：  ）
- 组合（composition）：实心菱形那一端表示 1..1（符号  ）

- 谓词模型（略）
- 信息世界与逻辑模型
  - 关系数据结构
    - 表结构
      - 二维表（简称“表”）：由表框架、表元组组成
      - 表框架：n 个命名的属性组成，n 为属性元数；每一个属性有取值范围，称为“值域”（数据库当中将 domain 翻译成值域，数学上应为定义域）
      - 表元组：一行数据，一个表框架可以存放 m 个元组，m 为表的基数
      - 二维表：n 元表框架及框架内的 m 个元组可以构成一个完整的二维表
    - 关系：满足七个性质（见 P33）的二维表称为**关系**
      - 关系模式：一个关系的关系和属性名的集合构成该关系的关系模式
      - 关系数据库模式：关系数据库中所有关系的关系模式的集合
      - 关系框架+关系元组=关系
      - 关系是关系模型的基本数据单位
    - 键/关键字（Key）：能**唯一标识关系中的一个元组的属性集**，每张二维表都至少存在一个“键”
      - 候选键/关键字：一个关系可以有多个关键字，也称候选关键字
      - 主键/关键字：从候选关键字中选出一个作为该关系的主关键字
      - 外键/关键字：表 A 的某**属性集**（注意此处：外关键字可以多个）取值来源于表 B 的主键，则该属性集即为 A 的外关键字。eg：学号、课程编号、成绩当中，学号和课程编号可以是外关键字
        - 注意：一个键不能成为不同表的外键，至多只能引用一张表中的主键!
        - 注意 2：引用自身表中的元组时，只能：正常取值或取空值（引用自己所在的元组时取空值）

【例】‘学生’表



【例】‘选课’表



        - 为什么能构成一个候选键？因为学号和课程号才能构成唯一性，一个学号可以选多门课，一门课也可以有多个学号选，所以只有把它们视为一个候选键才能确定键所需要的“唯一性”
- 计算机世界与物理模型
  - 提高文件 IO 操作效率的方法：索引、散列法（Hash）、集簇（Cluster）

关系数据库系统

- 关系数据库系统的衡量准则

- 空值：“无意义”或“当前未知”的值
  - 系统应有 n 能处理空值的能力
    - 空值的比较运算：不能参与比较运算
    - 含有空值对象的运算表达式计算：进行+、-、*、/的运算的结果均应该为空值
    - 含有空值的逻辑表达式，则其运算结果为逻辑假
    - 含有空值成员的集合上的统计运算：元组时空值可以不参与计算，列时空值应该考虑进来
    - SUM,AVG,MAX,MIN,COUNT 操作时集合的“空值”元素不统计在内
    - 统计计算中空集 SUM,AVG,MAX,MIN 的统计结果为空值；COUNT 统计结果为 0;
  - 数据完整性约束
    - 实体完整性：主键的属性不能有空
    - 参照完整性：外键要么空，要么是被引用表中当前存在的某元组上的主键值
    - 用户定义完整性：用户自己定义的属性取值约束

- 关系代数:
  - 概念:
    - 域（n 元关系 R 有 n 个域）：注意 n 元指属性元数而非表的基数，即多少列
    - 笛卡儿积
  - 关系模型上的数据操纵
    - 数据查询（选择->投影）:
      - 纵向定位（行选择，选择元组），然后横向定位（列指定，指定属性）;
      - 多表数据查询则在最开始先进行两表合并操作，逐渐合并两表至所有表都合为一张表
      - 分解为三种基本操作:
        - 两个关系的合并
        - 单个关系内的元组选择
        - 单个关系内的属性指定
    - 数据删除:（注意删除的是元组而非属性!）
      - 确定被删除的元组（注意一次删除只能删除**一个关系**内的元组，如果有多个关系，则需要执行多次删除操作）、执行删除操作
    - 数据插入:（注意插入的是元组而非属性!）
      - 一条数据插入只能向**一个关系**中增加新的元组
    - 数据修改：在一个关系中修改指定元组的某些分量上的值
      - 由其它数据操纵方式实现：删除元组+插入元组
  - 关系操纵的表示
    - 关系模型有四种操纵，可以分解为五种基本操作

关系上的五种基本操作	关系代数的五种基本运算
元组选择	选择运算
属性指定	投影运算
关系的合并	笛卡尔乘积
元组的插入	并运算
元组的删除	差运算

- 注意：修改=先删除+后插入，不算是基本操作
- 注意 2：交运算不是基本运算，可由差运算得：**R∩S = R – (R – S) = S – (S – R)**
- 投影运算：可以①略去关系中某些列②重新安排剩余列的次序
  - 运算符表示：$\prod_{A_{i1},A_{i2},\ldots,A_{im}}(R)$
  - 运算结果：是一个$A_{i1},A_{i2},\ldots,A_{im}$构成的 m 元关系
  - 投影必须要消除结果关系中可能会出现的重复元组

R

A	B	C
a	b	c
d	a	f
c	b	d

$\Pi_{C,A}(R)$

C	A
c	a
f	d
d	c

$\Pi_{B}(R)$

B
b
a

- 选择运算：根据给定条件 F 从关系 R 中选出符合条件的元组
  - 运算符表示：$\sigma_F(R)$
  - F = αθβ,(θ =</>/<=/>=/! =)和复合逻辑条件∧或∨
  - 字符时两端需加上单引号' '
- 笛卡尔乘积：
  - 设关系 R,S，分别为 n，m 元关系，分别有 p，q 个元组，则 R X S 关系为 **n+m** 元关系，有序元组个数为 **p*q**。注意到下面 B，C 列重合，因此需要加上 R.B，S.B 这种的东西

【例】

R

A	B	C
a1	b1	c1
a1	b2	c3
a2	b1	c2

S

B	C	D

R × S

R.A	R.B	R.C	S.B	S.C	S.D
a1	b1	c1	b1	c1	d1
a1	b1	c1	b1	c1	d3
a1	b1	c1	b2	c2	d2
a1	b1	c1	b1	c2	d4
a1	b2	c3	b1	c1	d1
a1	b2	c3	b1	c1	d3
a1	b2	c3	b2	c2	d2

- 元组插入 R∪R'
- 元组删除 R-R'
- 元组修改(R-R')∪R''
- 查询：单个关系选择+投影；多个关系笛卡尔乘积

- 数据查询：**先选择运算σ，再投影运算∏**，可以简写为∏σ(R) ：注意书写次序不能颠倒!!
- 投影运算不能交换、选择运算可以交换、投影和选择顺序必须是先选择再投影
  - 选择运算的交换律可用于查询优化，eg：先查年龄再查性别比先查性别再查年龄更好，并且结果不变
- 求最大?
  - 查编号->获知其它并非最大的->所有编号减掉非最大的，最后留下的就是最大的（最小、第二大等同理）
- 关系代数中的扩充运算
  - 交运算：∩
    - 条件：表头必须完全一样
    - 操作：选出既在 R 又在 S 中的有序元组（有序元组必须完全一样）

- 除运算：÷（笛卡儿积的逆运算）假设 T÷R=S 或 T/R=S
  - 条件：T 能被 R 除⇔T 中的域包含 R 中所有属性且 T 有一些域不在 R 中
  - 操作：S 的域由 T 中那些不出现在 R 中的域组成，其有序元组则由 R 中出现的所有元组在 T 中所对应的相同值组成。

R

A	B	C	D
1	2	3	4
7	8	5	6
7	8	3	4
1	2	5	6
1	2	4	2

S

C	D
3	4
5	6

R÷S

T

A	B
1	2
7	8

S

C	D
3	4

R÷S

T

A	B
1	2
7	8

S

C	D
3	4
5	6
4	2

R÷S

T

A	B
1	2

  - 注意：除法操作需要注意可能需要先投影运算，例如找出修读关系 C 中所有课程的学生的学号，首先需要投影到剩下学生学号和课程号 SC，然后 SC÷C 得到结果，不能直接除，因为可能有其他信息干扰到除法运算的结果。
  - 注意：if T=R÷S, then S×T 包含于 R
- 联接运算：$T=R \underset{F}{\bowtie} S$，其中 F=iθj，i 为 R 中的域，j 为 S 中的域，θ是比较操作符
  - 可以理解为笛卡儿积，然后选择满足 F 的元组：$R \underset{F}{\bowtie} S=\sigma_F(R\times S)$
  - 注意：联接运算中的同名属性不能消除，但是在结果关系中需要对同名属性换名，$Head(T)=Head(R)\cup_{all} Head(S)$

R

A	B	C	D
1	2	3	4
3	2	1	8
7	3	2	1

S

E	F
1	8
7	9
5	2

$R \underset{D>E}{\bowtie} S$

A	B	C	D	E	F
1	2	3	4	1	8
3	2	1	8	1	8
3	2	1	8	7	9
3	2	1	8	5	2

$R \underset{D=E}{\bowtie} S$

A	B	C	D	E	F
7	3	2	1	1	8

- 自然联接运算：$T=R \bowtie S$
  - 操作：根据两个关系中的同名属性（公共属性）进行等值联接
  - 注意：自然联接运算的同名属性应该合并，Head(T)=Head(R)∪Head(S)

R

A	B	C	D
1	2	3	4
1	5	8	3
2	4	2	6
1	1	4	7

S

D	E
5	1
6	4
7	3
6	8

$R \bowtie S$

A	B	C	D	E
2	4	2	6	4
2	4	2	6	8
1	1	4	7	3

R

A	B
a1	b1
a2	b2
a3	b5

S

B	C
b1	c1
b2	c2
b3	c3
b4	c4

R ⋈ S

A	B	C
a1	b1	c1
a2	b2	c2

- 关系代数实例：
  - 求解过程：查询目标->查询条件->确定从条件到目标的查找路径
  - 投影运算不到最后不要使用非关键字进行投影！

关系数据库语言 SQL

- 数据定义功能
  - 基表的定义：
    - <列定义>-> <列名><数据类型>[NOT NULL]
    - <基表> -> CREATE TABLE <基表名>(<列定义>{<列定义>})[其它参数]
  - 基表的修改：
    - 增加列：ALTER TABLE <基表名> ADD <列名><数据类型>
    - 删除列：ALTER TABLE <基表名> DROP <列名><数据类型>
  - 基表的删除：
    - DROP TABLE <基表名>
  - 索引的建立与删除
- 数据操纵功能
  - 数据（基本）查询：SELECT 可以在 WHERE 中嵌套
    - 目标子句: SELECT <列名>[, <列名>, …]
    - 范围子句: FROM <基表名>[, <基表名>, …]
    - 条件子句: WHERE <逻辑条件>
      - 这是与分组无关的条件
    - 分组子句: GROUP BY <列名>[, <列名>, …]
    - 分组查询子句: HAVING <组条件>
      - 这是分组的条件，区分 WHERE 命令
    - 排序输出子句: ORDER BY <列名>[ASC|DESC][, <列名>[ASC|DESC], …]，缺省升序。
    - 例子：查询成绩全都及格（WHERE）的同学的平均成绩（GROUP BY），输出其中某些列（HAVING），如果要排序，那么 ORDER BY
    - 常用谓词
      - DISTINCT：去重
      - BETWEEN：在…和…之间（两端取等）
      - LIKE：模式匹配，<列名> IS [NOT] LIKE <字符串常量>[ESCAPE val2]
        - ESCAPTE val2 表示 val2 后面的_和%不再作为通配符而是作为这个字符本身被识别
      - NULL：判断值是否为空值
        - 注意区别='NULL'，这个表示格子里面填的是否是四个字母'N', 'U', 'L', 'L'
    - 布尔表达式：优先级：NOT>AND>OR，可能需要另外加括号
  - 分层结构查询与集合谓词使用

- expr [NOT] IN (subquery)：标量与集合量的属于比较
- expr θ ANY|ALL (subquery)：限定比较谓词，标量与集合中元素的比较（ANY：任意一、ALL：所有）
- [NOT] EXISTS (subquery)：是否为空集的判断谓词

- SELECT 语句之间的运算：不加 ALL 表示集合运算，消除相同元素；加 ALL 则不消除相同元素
  - UNION [ALL]：并运算
  - INTERSECT [ALL]：交运算
  - EXCEPT [ALL]：差运算
- SQL 计算、统计、分类的功能
  - 统计功能：COUNT、SUM、AVG、MAX、MIN
    - 注意：不能再 WHERE 子句当中直接使用统计函数！只能在 SELECT 语句当中使用
    - 注意：空值的统计计算将直接被忽略
    - 注意：空集的统计运算 COUNT()返回 0，其它统计函数返回空值 NULL
  - 计算功能：+、-、*、/、()、……
  - 分类功能：
    - GROUP BY <列名>[, <列名>, …]：对 SELECT 语句得到的集合元组分组，可以分别进行统计计算，实现分类统计查询
      - 注意是按照列名的取值的不同来划分集合
      - 注意 GROUP BY 只会输出一个元组，如果查询导致可能分类输出多个元组，那么一定错了！

    ❑ 一个很重要的事情是需要保证出现在**select**语句中但没有被聚集的属性只能是出现在**group by**子句中的那些属性

    ❑ 换句话说，任何没有出现在**group by**子句中的属性如果出现在**select**子句中的话，它只能出现在聚集函数内部，否则这样的查询就是错误的

    ❑ 例如，下面的查询是错误的

    ```
    SELECT dept_name, ID, avg( salary )
    FROM instructor
    GROUP BY dept_name
    ```

    ➢ 在一个特定分组（通过**dept_name**定义）中的每位教师都有不同的**ID**，每个分组只能输出一个元组，无法确定输出哪个

    - HAVING：设置逻辑条件
- 相关子查询执行过程：
  - 从外层查询中取出一个元组，将元组相关列的值传给内层查询。
  - 执行内层查询，得到子查询操作的值。
  - 外查询根据子查询返回的结果或结果集得到满足条件的行。
  - 然后外层查询取出下一个元组重复做步骤 1-3，直到外层的元组全部处理完毕。

- 数据控制功能【待补充】
- 数据交换功能【待补充】
- 拓展功能【待补充】

- 习题课

数据库的安全性与完整性保护：会考语句怎么写！

- 数据库安全的基本概念与内容：知道即可
  - 可信计算基（TCB）：为实现数据库安全的所有实施策略与机制的集合，是实施、检查、监督数据库安全的机构，是一个抽象的概念。
  - 主体、客体与主客体分离
    - 主体：数据库中数据的访问者
    - 客体：数据库中的数据及其载体，如表/视图/快照/存储过程/数据文件等
    - 主体集单向访问客体集
    - 数据库安全的基础：
      - 有关实体的主客体划分
      - 主客体之间的访问关系的控制
  - 主体访问客体存在三种安全控制与检查的控制方式：
    - 身份标识与鉴别：最外层、最简单、最基本的安全控制方式
      - 每一主体有自己的标识符，如用户名、口令
      - 常用的控制措施：登陆控制、口令字选取、对鉴别数据的保护、会话连接挂起、用户账号和属性文件
    - 自主访问与控制（DAC）：主体访问客体的常用安全控制方式，适合单机
      - 存取矩阵：主体、客体、存/取操作

	主体1	主体2	……	主体i	……
客体1	……	……	……	读/写	……
客体2	……	……	……	读/修改	……
……	……	……	……	……	……
客体j	插入	修改/删除	……	读/插入/删除	……
……	……	……	……	……	……

存取矩阵模型图

      - 基于存取矩阵的自主访问控制：安全政策范围验证、控制主体访问客体的规则验证、超越 DAC 的特权验证
    - 强制访问控制（MAC）：主体访问客体的弈中强制性的安全控制方式，用于网络环境，对网络中的数据库安全实体做统一的强制性的访问管理
      - 主客体标记：
        - 安全级别标记：规定了主客体的安全级别
        - 安全范围标记：规定了主体可以访问的范围、客体所处的范围
      - 访问时主体界别与客体级别满足比较关系的时候才能允许访问
      - Bell-Lapadula 模型：
        - 下读、上写，见书 P87
        - 由专门的安全管理员设置，任何主体均无权设置与授权。
      - 强制性的访问控制措施：安全政策范围验证、验证访问控制和信息流动的原则、验证可超越 MAC 特权
  - 数据完整性：防止非法使用插入（INSERT）、删除（DELETE）、修改（UPDATE）

等影响数据完整性的操作
    - 常用控制手段：三类数据完整性
        - 实体完整性：例如主键
        - 关联完整性：例如引用时原来被引用的要存在
        - 用户定义完整性约束：例如人年龄不能达到负数或上千岁
- 隐蔽通道：非法访问通道，不受 TCB 控制，应该予以防止
- 数据库安全的形式化模型：对数据库安全模型的安全策略作形式化描述证明
- 审计：跟踪记录用户对数据的访问操作并可能可以根据审计结果给出报警信息
    - 访问时间、类型、访问客体名、是否成功、结果……
    - 审计有关的保护措施：审计事件控制、入侵检测与应对、审计记录的保护、审计记录的分析/查阅
- 访问监控器：独立的、最小的、防篡改的自主机构，用以监控主体和客体的授权访问关系
    - TCB 抽象的功能策略集合，但访问监控器真实存在，是 TCB 在网络中的实现

- SQL 对数据库安全的支持：C1 级别
    - 主体、客体及主客体分离
    - 身份标识与鉴别
    - 数据完整性
    - 自主访问控制与授权功能
        - （用户，操作对象，操作权限）的三元组定义
        - 授权语句：GRANT <操作权限列表> ON <操作对象> TO <用户名列表> [WITH GRANT OPTION]（可不可以把自己的权限同样给别人）
            - GRANT SELECT, UPDATE ON S TO XULIN WITH GRANT OPTION
                - 表示将表 S 上的查询域修改权限授予用户 XULIN，同时该用户 XULIN 还可以将所获权限传递给其它用户
        - 回收语句：GRANT <操作权限列表> ON <操作对象> FROM <用户名列表> [RESTRICT|CASCADE][不存在连锁回收问题时才能回收权限，否则拒绝回收|连锁回收]
            - 连锁回收的例子：回收张三时，张三授权给李四的权限也回收
            - REVOKE UPDATE ON S FROM XULIN CASCADE
                - 表示从用户 XULIN 中回收表 S 上的修改权，并且是连锁回收
- 数据库的完整性
    - 指数据库中数据的正确性（有效性）和一致性（并发访问时，保证对数据的更新不会出现与实际不一致的情况）
    - 数据库完整性保护的功能
        - 设置功能、检查功能、处理功能
    - 完整性规则的三个内容
        - 实体完整性规则（主键属性取值不能为空）、参照完整性规则、用户定义的完整性规则
    - 完整性约束的设置、检查与处理
        - 完整性约束条件的设置
            - 属性级约束：域约束
                - 一些需要考虑的问题
                    - NOT NULL vs. DEFAULT NULL：区分哪些列非空，哪些列默认

空
- Constraint name：主关键字、外关键字可以考虑取名字，可选
- UNIQUE vs. NOT NULL：区分唯一键和主键？唯一键如果取值一定要是唯一的，且可以是空值；但是候选键、主键则唯一且不可能是空值。如果 UNIQUE+NOT NULL，那么认为这个属性可以作为一个候选键
- PRIMATY KEY vs. NOT NULL：具体问题具体分析，例如姓名不能作为主键，但是非空等等
- REFERENCES：外键一般用 RESTRICT 属性
- CHECK：属性级约束

- 元组级约束：表约束
  - 候选键
  - 外键
  - 检查
- 全局约束：断言 assert（不一定支持）

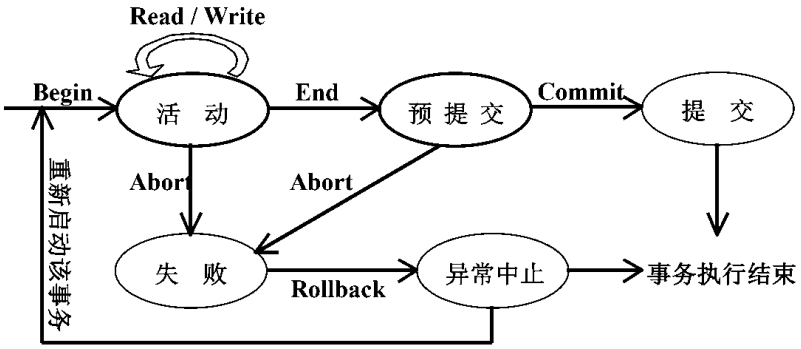
- 完整性约束条件的检查
- 完整性约束条件的处理

- 触发器：一个事件的发生会导致另外一些事件的发生——触发器
  - 触发事件（用户定义）给出触发条件，触发（DBMS 检测）的话就调用相应的结果事件（用户定义）进行处理

事务处理、并发控制与故障恢复技术

- 事务处理
  - 定义：由某个用户所执行的一个不能被打断的对数据库的操作序列
  - 事务用来保持数据库中数据的一致性、用户对数据库操作的连贯性
  - ☆☆☆事务的 ACID 特性：**A**tomicity、**C**onsistency、**I**solation、**D**urability
    - 原子性（Atomicity）：不可分割的操作序列，要么全执行，要么全不执行
      - 事务管理子系统、事务日志，DBMS 会自动维护用户事务执行的原子性
    - 一致性（Consistency）：事务执行结果令数据库由一种一致性达到新的一致性
      - 状态：数据库中所有数据对象当前的取值情况
      - 基于的假设：
        - 事务开始前数据库处于一致状态
        - 没有“其他事务的干扰和系统故障”
      - 事务的一致特性两方面完成
        - DBMS 中的“数据完整性保护”子系统
        - 编写事务的应用程序员完成
    - 隔离性（Isolation）：多个事务并发执行时一个事务不必关心其他事务的执行
      - 由 DBMS 的并发控制子系统来实现的
    - 持久性（Durability）：一个事务一旦完成全部操作，它对数据库的更新永久反映在数据库中，即使以后发生故障，也应保留事务执行的结果
      - 由 DBMS 的恢复管理子系统实现
  - 事务活动
    - 事务夭折：Abort

- 事务回滚：Rollback
- 事务提交：Commit



事务的状态变迁图

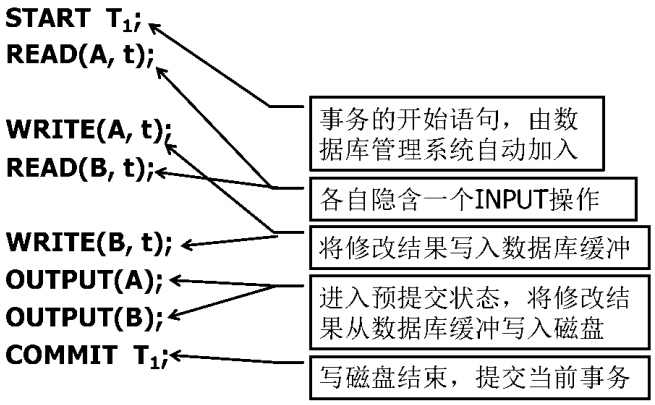
- 注意：无论事务是提交状态还是异常中止状态，都意味着一个事务执行结束
- 与事务有关的控制语句：
  - BEGIN TRANSACTION
    - 数据定义命令 DDL、打开自动提交、数据操纵命令 DML
  - COMMIT TRANSACTION
  - ROLLBACK TRANSACTION（可选检查点）
  - 三种与事务有关的控制命令 DCL：
    - 设置事务的自动提交 SET AUTO COMMIT ON|OFF
    - 设置事务类型：SET TRANSACTION READONLY|READWRITE
    - 设置事务的隔离级别：SET TRANSACTION ISOLATION LEVEL ……

- 常见的并发执行错误
  - 丢失修改：对多个事务并发修改同一个数据对象的情况未加控制
  - 脏读：一个事务读取另一个事务未提交的修改结果
  - 不可重复读：在两次读操作之间插入了另一个事务的写操作
- 有关事务的语句
  - SET TRANSACTION ISOLATION LEVEL [控制级别]
  - 控制级别
    - 未提交读：READUNCOMMITED
    - 提交读：READCOMMITED
    - 可重复读：READREPEATABLE
    - 可序列化：SERIALIZABLE
  - 一般来说，写的时候排它写直至事务结束

	Dirty read	**Non-repeatable read**	**Phantom read**
Read uncommitted	Possible	Possible	Possible
Read committed	Not possible	Possible	Possible
Repeatable read	Not possible	Not possible	Possible
Serializable	Not possible	Not possible	Not possible

- 事务的组成
  - 数据对象
  - (cont.)

- 操作
  - 事务控制
    - 事务开始：START T_0
    - 提交事务：COMMIT T_0
    - 回退/放弃事务：ABORT T_0
  - 数据访问：关键的操作 READ、WRITE、OUTPUT
    - 磁盘数据对象 A 读入内存缓冲区：INPUT (A)
    - 内存缓冲区数据对象 A 写入磁盘：OUTPUT (A)
    - 内存缓冲区数据对象 A 读入内存 t：READ (A, t)
      - 可能隐含一个 INPUT 操作，即 INPUT 操作可以被忽略
    - 内存 t 写入内存缓冲区数据对象 A：WRITE (A, t)
  - 数据库日志



- 并发控制技术
  - 事务的并发执行
    - 执行方法
      - 串行执行、并发执行、并发执行的可串行化
    - 概念
      - 调度、串行调度
      - 可串行化调度：一个调度对数据库状态的影响和某个串行调度相同，那么这个调度是可串行化调度
      - 事务及其调度的表示方法：事务 T_i 读/写数据库对象 X 表示为 $r_i(X)$和 $w_i(X)$
      - 冲突可串行化：
        - 冲突：调度中一对连续操作(op1;op2)如果满足交换两者的执行顺序，那么涉及的事务至少有一个的行为会改变，那么这对连续操作是冲突

          ❑ **冲突的判断**

          ➢ 假设T_i和T_j是两个不同的事务(即i≠j)，则：

          ▪ $r_i(X);r_j(Y)$ 不是冲突(即使 X=Y 也不会是冲突)

          ▪ 如果 $X \neq Y$，则 $r_i(X);w_j(Y)$ 不会是冲突

          ▪ 如果 $X \neq Y$，则 $w_i(X);r_j(Y)$ 不会是冲突

          ▪ 如果 $X \neq Y$，则 $w_i(X);w_j(Y)$ 不会是冲突

          ➢ 同一事务的任意两个相邻的操作都是冲突，如：

          $(r_i(X);w_i(Y))$、$(w_i(X);r_i(Y))$、$(r_i(X);r_i(Y))$等

          ➢ 不同事务对同一数据对象的‘写’冲突：

          $w_i(X);w_j(X)$

          ➢ 不同事务对同一数据对象的‘读-写’冲突：

          $r_i(X);w_j(X)$　　$w_i(X);r_j(X)$
        - ■

        - 可串行化调度：初始给定的调度能够通过一组“非冲突”操作的交换，能够将该调度转换成一个串行调度
        - 注意：冲突可串行化是可串行化的充分不必要条件
            - 冲突可串行化调度一定是可串行化调度
            - 可串行化调度并不一定是冲突可串行化的
        - 优先图
            - 如果事务优先图无环，那么调度 S 就是冲突可串行化的；否则，若事务优先图有环，那么调度 S 就不是冲突可串行化的调度。
    - 常见的并发执行错误：
        - 丢失修改：T1 读->T2 读->T1 修改，写回磁盘->T2 修改写回磁盘->T1 的修改被丢失
        - 脏读：T1 读完->T1 写回磁盘->T2 读->T1 撤销->T2 获得脏数据
        - 不可重复读：T1 读->T2 读，写->T1 验算结果->T1 验算结果不一致
        - 主要产生原因：违反了事务 ACID 四项原则，特别是隔离性原则
- 封锁
    - 封锁的作用：由封锁的类型决定和由系统所采用的封锁协议决定
    - 锁的类型：排他锁（X 锁）和共享锁（S 锁）
        - 排他锁：加锁的事务可以读写数据对象，而其它事务被禁止访问
            - 加锁条件：数据对象没有被任何事务封锁
            - 缺点：降低系统的并行性
            - 特点：事务 T 独占数据对象 A，且 X 锁必须维持到事务 T 的结束
        - 共享锁：加锁的事务可以读数据对象，但不能写，其它事务仍可以加 S 锁
            - 加锁条件：数据对象没有被任何事务封锁，或者其它事务仅仅以 S 锁封锁数据对象
            - 特点：
                - 数据对象的所有 S 锁释放之前，不能被写
                - S 锁不必维持到事务的结束，提高系统的并发性
    - 封锁的相容矩阵

		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（well-formed）事务：大概就是按规矩申请释放锁，保证并发事务正确执行的基本条件
- 三级封锁协议
    - 一级封锁协议：写操作前加 X 锁直到事务结束
        - 防止丢失修改
    - 二级封锁协议：读操作前加 S 锁，读完释放+一级封锁协议
        - 防止丢失修改、脏读
    - 三级封锁协议：读操作前加 S 锁，事务结束释放+一级封锁协议
        - 防止丢失修改、脏读、不可重复读
        - 注意：三级封锁协议不是一级+二级+三级而是一级+三级！
    - 注意：三级封锁协议和两阶段封锁协议不是一回事！前者是封锁规则，后者是

执行阶段!

		可能出现的数据不一致现象		
		丢失修改	脏读	不可重复读
封锁协议	一级封锁协议	**No**	**Yes**	**Yes**
	二级封锁协议	**No**	**No**	**Yes**
	三级封锁协议	**No**	**No**	**No**

- 两阶段封锁协议（2PL 协议==拓展==>2PL 事务）
  - 第一阶段/扩展阶段：申请并获得封锁
  - 第二阶段/收缩阶段：释放所有申请获得的锁
- 封锁粒度
  - 事务封锁的数据目标的大小
  - 封锁对象：
    - 逻辑数据单元
      - 属性值（集合），元组，关系
      - 索引项，索引文件
      - 整个数据库
    - 物理数据单元
      - 页、块等
  - 封锁粒度和系统并发度/并发控制开销的关系

封锁粒度	系统并发度	并发控制的开销
大	低	小
小	高	大

  - 多粒度封锁
  - 意向共享锁、意向排它锁、共享意向排它锁
- 活锁与死锁
  - 活锁：某些事务永远处于等待状态，得不到解锁机会
    - 解决方法：先来先解决。
      - 例如已经有共享锁，那么排他锁申请不了，如果一直有共享锁进来，那么排他锁就陷入活锁，永远无法执行。这里可以让后来的共享锁事务排在排他锁事务后面（先来先服务），避免活锁。
  - 死锁：事务之间对锁得循环等待
  - 死锁的处理方法
    - 预防法
      - 顺序申请法
      - 一次申请法
    - 死锁解除法
      - 超时死锁检测法：一般数据库用的就是超时死锁检测法，报一个超时错退出

        - 事务的执行时间超时
        - 锁申请的等待时间超时
      - 等待图法
      - 时间戳死锁检测法
- 数据库恢复技术
  - 概述
  - 数据库故障分类
    - 小型故障：影响范围处于一个事务之内
      - 逻辑错误：例如数据输入错误、数据溢出、资源不足
      - 系统错误：例如死锁、事务执行失败等
    - 中型故障：磁盘数据不受到影响、数据库不受到破坏
      - 系统故障：如系统硬件故障、操作系统、DBMS、应用程序代码等。
      - 外部故障：外部原因（如停电）引起的
    - 大型故障
      - 磁盘故障
      - 计算机病毒
      - 黑客入侵
  - 数据库故障恢复三大技术
    - 数据转储：定期将数据库中的内容复制到其它存储设备中的过程
      - 又称后援副本或备份
      - 动态转储、静态转储；海量转储、增量转储
      - 动态转出需要用到日志记载
        - 转储的开始点和结束点
        - 转储执行过程中事务的执行状况
          - <事务标识，更新对象，更新前的值，更新后的值>
        - 在转储的执行过程中完成的事务的结束状态
          - Commit、Abort
    - 日志：此处一般对修改型操作记日志，对读操作不记
      - 内容
        - 每个更新操作的事务标识、更新对象、更新前和/或更新后的值
        - 每个事务的开始、结束等执行情况
        - 其它信息
      - 作用
        - 确保事务执行的原子性
        - 实现增量转储
        - 实现故障恢复
      - 几种不同的日志
        - Undo 日志：放弃事务的撤销
          - 开始事务：<Start T>
          - 提交事务：<Commit T>：磁盘全部写完写该日志
          - 放弃事务：<Abort T>
          - 更新记录：<T, X, V>：X 的新值写到磁盘前写该日志
            - 事务 T 修改了数据库元素 X 的旧值 V
          - 看 ppt：五种不同情况的故障恢复分别怎么分析？

- 思考题：如果在恢复过程中再次出现系统崩溃？如何进行系统的故障恢复？
  - 无论到什么地方崩溃，只要重新恢复一遍即可
- 检查点：减小故障恢复的开销，<CKPT>

- Redo 日志：已提交事务的重做
  - 开始事务：<Start T>
  - 提交事务：<Commit T>
  - 放弃事务：<Abort T>
  - 更新记录：<T, X, V>
    - 事务 T 修改了数据库元素 X 的**新**值 V：和 Undo 日志的区别
  - 一定是先写日志在磁盘上，才可以写数据到磁盘
  - Redo 日志的故障恢复过程？
- 考试考过：redo 日志和 undo 日志的区别

redo日志与undo日志的主要区别

1) 恢复的目的不一样
2) 提交记录**<Commit T>**写入日志的时间不一样
   - **undo**日志：在事务**T**的所有数据库磁盘修改操作**(Output)**结束后才能在日志中写入提交记录**<Commit T>**
   - **redo**日志：在提交记录**<Commit T>**被写入磁盘后才能将事务**T**更新后的值写入数据库的磁盘中
3) 在更新记录**<T,X,V>**中保存的值**V**不一样

- undo 与 redo 日志的不足。（没说过考试考过）

undo与redo日志的不足

- **Undo**日志要求数据在事务结束后立即写到磁盘，可能增加需要执行磁盘**I/O**的次数
- **Redo**日志要求事务提交和日志记录刷新之前将所有修改过的数据保留在内存缓冲区中，可能增加事务需要的平均缓冲区的数量
- 如果被访问的数据对象**X**不是完整的磁盘块，那么在**undo**日志与**redo**日志之间可能产生相互矛盾的请求

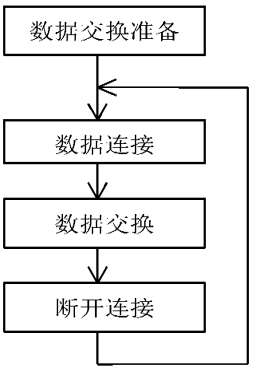
- 事务的撤销与重做：UNDO 和 REDO

- 恢复策略：小型、中型、大型
- 数据库镜像

数据库中的数据交换

- 概述
  - 数据交换是数据主体（数据的使用者）与数据客体（数据库）之间的数据交互过程
  - 数据交换的五种方式
    - 初级阶段与人-机交互方式
    - 中级阶段

        - 嵌入式方式
        - 自含方式
        - 调用层接口方式
    - 近期阶段与 Web 方式
- 数据交换的管理
    - 会话管理：会话的进行须预先作环境的设定
        - 设定的环境参数：数据客体模式、语言模式、时间模式、标识符
    - 连接管理：负责在数据主、客体之间建立实质性的关联
    - 游标管理：将 SQL 变量中的集合型变量逐个取出后送入应用程序变量供其使用
        - 增加游标（curser）语句
            - 定义、打开、推进、关闭
    - 诊断管理:生成、获取诊断值(数据主体发出数据交换请求后数据客体返回两种值:数据值和执行状态值（也称诊断值)）的管理
        - 诊断区域和诊断操作
    - 动态 SQL
        - SQL 语句正文动态、变量个数动态、类型动态、SQL 语句引用对象动态
        - 准备语句、执行语句、立即执行语句
- 数据交换的流程



- 数据交换的四种方式
    - 交互式 SQL：独立运行
    - 嵌入式 SQL：SQL+主语言
        - 对嵌入在主语言中的 SQL 语句加前缀 EXEC SQL 和后缀 END_EXEC 或;
    - 自含式 SQL：可在数据库服务器中独立运行，常用于编写存储过程、存储函数、触发器
    - 调用层接口
    - Web 方式

数据库的物理组织

4. 磁盘存储器的I/O操作

- 编码方式：设一个磁盘存储器有n个柱面，每个柱面有m个磁道，每个磁道有r个磁盘块。则编码规则如下：
  - ❖柱面号：由外层到内层分别为（0号柱面，1号柱面，……，n-1号柱面）
  - ❖磁道号：每个柱面中的磁道从上到下分别为（0号磁道，1号磁道，……，m-1号磁道）
  - ❖磁盘块号：每个磁道中的磁盘块按照旋转的方向分别为（0号磁盘块，1号磁盘块，……，r-1号磁盘块）
- 因此，该磁盘存储器中共有m*n*r个磁盘块，其中x号柱面的y号磁道的z号磁盘块的编号是：

  x * m * r + y * r + z

➢ 常用的磁盘块的分配方式

	分配方式	优点	缺点
连续分配法	按连续物理地址分配	有利于文件的顺序访问	无法扩充文件所使用的存储空间
链接分配法	随机分配，磁盘块之间通过指针链接在一起	有利于文件存储空间的扩充	存取效率较差
索引分配法	通过一个逻辑块号与磁盘块地址之间的索引来维护所使用的磁盘块	空间的使用较灵活	索引文件需要额外的存储空间
集簇分配法	局部是物理上连续的，又可以通过指针链接新的磁盘块	综合了连续分配法和链接分配法的优点	

- 索引技术
  - 索引的优点?
    - 通过建立索引文件可以大大提高在数据文件上的记录查找定位速度
      - 索引文件的大小一般都大大小于数据文件的大小
      - 索引文件采用便于查找的特殊组织结构
    - 在不同类型的数据文件上我们可以建立不同的索引文件
  - 稠密索引
    - 索引文件
      - 关键词：关键字值、指向记录本身的指针、按照关键字值排序
    - 稠密索引定义：每条记录在索引文件中都存在一个相对应的索引项
    - 查找算法：二分查找关键字索引项，不存在则失败，根据索引项的记录指针到数据文件中定位读取
  - 稀疏索引
    - 查找算法：二分查找关键字值≤K 的索引项，若找不到则失败，若找到则根据索引项的记录指针到数据文件中读取相应磁盘块，最后在磁盘块中二分查找关键字值为 K 的记录

- 稠密索引 与 稀疏索引 的区别
  - 索引文件的定义不同
    - ❖稀疏索引只能用于顺序文件上的索引组织
    - ❖稠密索引中的每个索引项对应数据文件中的一条记录，而稀疏索引中的每个索引项则对应数据文件中的一个磁盘块
  - 需要的磁盘空间大小不同
  - 在记录的查找定位功能上存在差别：
    - ❖稠密索引：可以直接回答是否存在键值为K的记录
    - ❖稀疏索引：需要额外的磁盘I/O操作，即需要将相应的数据文件中的磁盘块读入内存后才能判别该记录是否存在

- 多级索引:
  - 注意：一级索引是直接建立在数据文件上的索引，一级索引可以是稠密索引或稀疏索引；二级及以上索引是稀疏索引
    - **索引文件本身是一个顺序文件，利用索引文件或多级索引可以大大提高数据文件的访问效率**
    - **索引顺序文件的不足**
      - ❖**记录查找算法的效率不高（$\log_2 N$）**
      - ❖**在数据文件是非顺序文件，或索引关键字值的变化比较频繁的情况下，索引顺序文件自身的维护非常复杂，对索引项的插入、修改和删除操作会导致索引项在索引文件中的大量移动**
      - ❖**在上述情况下，如果通过引入链接磁盘块的方法来减少索引项的移动，又会减低存储空间的利用率，并最终影响到系统的性能**
      - ❖**因此需要引入适合于索引文件的存储组织的文件结构，这就是在数据库系统中广泛使用的多级索引技术：B/B+树索引**
- B/B+树
  - 平衡性、过半性、顺序性、自适应性
  - B+树：知道大概怎么回事，怎么查就可以了
    - 效率
      - 随机查找的效率：每次所需要的磁盘 I/O 次数等于 B+树的高度
      - 空间利用率：>50%
      - B+树上的插入、删除操作效率是 Logn 复杂度
  - B 树和 B+树的区别?
    - 组织方式不一样
      - ❖ **B+树：所有有效的索引关键字值都必须存储在叶结点中，其内部结点中的键值只用于索引项的查找定位。**
      - ❖ **B树：有效的索引关键字值可以出现在B树的任意一个结点中。**

- 内部结点不同
  - ❖B^+树：关键字值 + 子树指针
  - ❖B树：关键字值 + 记录指针 + 子树指针
    - 因此B树结点的扇出（*即一个结点可以拥有的最大子结点数目*）较小，从而导致整个B树的高度大于B^+树的高度。

- 随机查找效率的区别
  - ❖B^+树：所有关键字的查找速度基本一致
  - ❖B树：依赖于关键字所在结点的层次

- 在B树中没有提供对索引关键字的顺序扫描功能。

- B树的插入、删除操作较B^+树复杂。

- 数据库与文件

关系数据库规范化理论

- 概述
  - 数据冗余度大
  - 插入异常
  - 删除异常
  - 关系的规范化
    - 属性与属性之间的内在语义联系函数依赖&多值依赖
    - 关系的规范化：每个关系中，属性和属性之间一定要满足某种内在的语义联系
    - 范式：关系的规范化根据属性间存在的内在语义联系要求的不同分为若干级别
- 规范化理论
  - 规范化的途径：
    - 竖向规范化：切属性集，采用投影和联接运算，形成成熟的规范化理论：模式分解理论（无损联结性、依赖保持性）
    - 水平规范化：切元组集合，采用选择和并运算，尚未形成成熟的规范化理论
  - 函数依赖
    - X 确定，Y 必确定，则 Y 函数依赖于 X/X 函数决定 Y，记 X->Y
    - X 称决定因素，Y 称依赖因素
    - 即若 X 取值相等，Y 取值必相等且唯一

T1

A	B
x1	y1
x2	y2
x3	y1
x4	y1
x5	y2
x6	y2

A→B？ √

B→A？ X

T2

A	B
x1	y1
x2	y4
x1	y1
x3	y2
x2	y4
x4	y3

A→B？ √

B→A？ √

T3

A	B
x1	y1
x2	y4
x1	y1
x3	y2
x2	y4
x2	y3

A→B？ X

B→A？ √

- 函数依赖一些概念
  - 平凡/非平凡的函数依赖
  - 完全函数依赖/部分依赖
  - 间接依赖关系/直接依赖关系
- Armstrong 公理系统
  - 要会用！以后会反复用！
  - 自反规则、增广规则、传递规则
  - 分解规则、合并规则、伪传递规则
  - FD 逻辑蕴涵
  - 函数依赖集 F 的逻辑闭包：F+
- 关键字
  - 完全函数依赖且 K 是 U 的子集
- 主属性集
  - 包含**所有关键字**，注意不要漏掉任何关键字！老师上课提醒了
- 非主属性集
  - 所有非键（关键字）的属性所构成的集合，注意非主属性集可以为空，但主属性集一定不能为空！
- 关键字与属性集闭包的关系？
  - $K_F^+ =$ U且对于 K 的任意真子集 Z，都有$Z_F^+ \neq U$
- 题目：先求关键字、求范式、求正规范化
  - 求关键字？Armstrong 公理系统或用属性集闭包循环算法
    - 注意一定可能存在多个关键字的情况！不能漏掉关键字！不然就全错了

- 与函数依赖有关的范式
  - 第一范式 1NF
    - 关系中的每个属性值都是一个不可分割的数据量（即不能是集合）
  - 第二范式 2NF
    - R∈1NF 且每个非主属性完全函数依赖于键/关键字
    - 判断一个关系模式是否满足 2NF 的要求？
      - 找出该关系模式的所有（候选）关键字
      - 据此确定该关系的主属性集和非主属性集
      - 判断每一个非主属性和关键字之间的函数依赖关系是否满足 2NF 的要求
      - 如果不存在非主属性对关键字的部分依赖，那么 SCG∈2NF
    - 要记得结合 ppt 上面的例子理解
  - 第三范式 3NF
    - 基于 2NF 定义，且每个非主属性既不部分依赖也不传递依赖于键
  - BCNF 范式：
    - 基于 1NF 定义，且每一个 X->Y 都有 X 必含键
    - BCNF 一定推出 3NF，但是 3NF 不一定推出 BCNF
- 多值依赖与第四范式
  - 特点：
    - 一个 X 一经确定，有一组 Y 与之对应
    - X 一经确定，对应的一组 Y 值与 U-X-Y 无关

- 记号：X->->Y
- 平凡的多值依赖（Z=U-X-Y = 空集）、非平凡的多值依赖（Z=U-X-Y≠空集）
- 性质
  - 函数依赖是一种多值依赖（特殊的值个数为 1 的多值依赖），但是多值依赖就不是函数依赖
  - if X->->Y，then X->->U-X-Y
- 多值依赖一些运算规则可以知道，不要求掌握

- 第四范式：若 X->->Y 是平凡的多值依赖，则 X 必含有关键字，此时称关系模式 R 满足第四范式，并记作 R∈4NF
  - 特点：
    - 只允许出现平凡多值依赖
    - 函数依赖要满足 BCNF

- 规范化所引起的一些问题
  - 函数依赖理论的研究
    - **函数依赖集的等价**：当且仅当两个函数依赖集的闭包相等
      - 判断方法：两边可以用 Armstrong 公理系统互相推导就代表二者等价
      - 判断方法 2：也可以通过**属性集闭包相等**的计算算法证明二者等价（更简单一点）
    - **最小函数依赖集**（最小覆盖）：与函数依赖集等价的所有函数依赖集中最小的
      - 判断方法：（其中条件一为了方便条件二三的判断，非必需）

        1. 依赖因素 A 为单个属性；
        2. 令：$F_1 = F - \{ X \rightarrow A \}$, 则 $F_1^+ \neq F^+$；
           - 不存在冗余的函数依赖关系
        3. 对于决定因素X的每一个真子集$Y(Y \subset X)$ 均作如下判断：令 $F_2 = F - \{X \rightarrow A\} \cup \{Y \rightarrow A\}$, 则 $F_2^+ \neq F^+$
           - 不存在部分函数依赖关系

        - 如果 F 中的每一个 FD 关系 X->A 均符合上述三个条件的要求，则 F 是一个最小函数依赖集
      - 算法：
        - 将依赖因素均改写为单个属性
        - 消去冗余的函数依赖
        - 消去部分函数依赖（合并）
  - 模式分解的研究：无损联接性，依赖保持性
    - 无损联接性：分解后，原关系中的信息不会被丢失
      - 每一个分解的关系的自然联结和原来的关系相同，就满足无损联接性
      - 如果分解的关系的自然联结比原来的关系多出来非空的元组，那么就不满足无损联接性

        定理8.3.2.1：如果R的分解为 $\rho = \{R_1, R_2\}$，F为R所满足的函数依赖集合，分解ρ具有无损联接性的充分必要条件是：

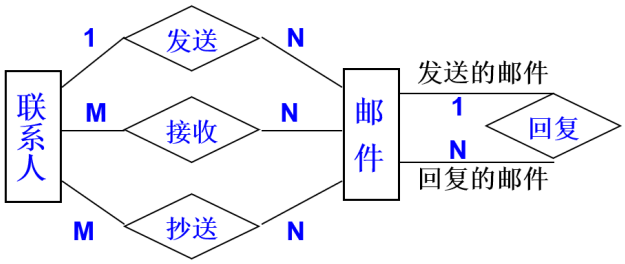
      - $R_1 \cap R_2 \rightarrow (R_1 - R_2)$ 或 $R_1 \cap R_2 \rightarrow (R_2 - R_1)$

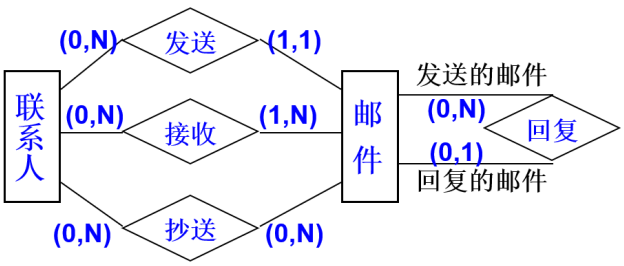
- 依赖保持性：原有的函数依赖关系在分解后的关系模式上依然存在
  - 分解完所有关系的并集的函数依赖集的闭包和原来函数依赖集的闭包相同，则满足依赖保持性
- 将关系 R 分解到第三范式并且该分解满足无损联接性和依赖保持性
  - 算法
    - 先计算函数 F 的最小依赖集
    - 。。。什么的

数据库设计

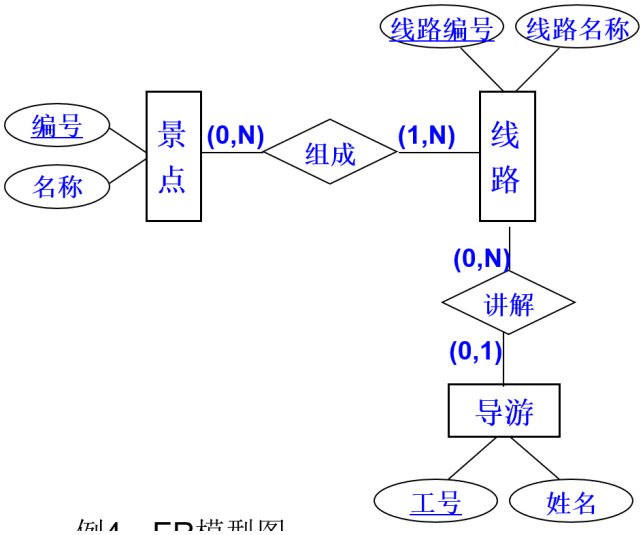
- 数据库设计概述
  - 信息需求：用户的数据、结构及其要求
  - 处理需求：用户对数据的处理过程和方式
  - 数据模式：概念模式、逻辑模式和物理模式
  - 数据库设计的四个阶段：
    - 需求分析--需求分析说明书->概念设计—概念设计说明书->（DBMS 模型限制）逻辑设计—逻辑设计说明书->（网络、硬件及系统软件平台限制）物理设计—物理设计说明书->
- 数据库设计的需求分析
  - 需求调查
  - 需求分析
    - 数据边界
    - 绘制数据流图（DFD）
    - 创建数据字典
      - 数据项、数据结构、数据流、数据存储、数据处理
      - 注意这里只是搞清楚要做什么，还没有到怎么设计编码细节（如具体数据类型如何在数据库中表示）的阶段，只要搞清楚做什么就行
  - 生成数据需求分析说明书
- 数据库的概念设计
  - 数据库概念设计概述
    - 目的：建立一个概念数据模型
    - 工具：E-R 模型、EE-R 模型、面向对象方法等
    - 方法：集中式模式设计法、视图集成设计法
  - 数据库概念设计的过程
    - 用户分解
    - 视图设计
      - 设计次序
        - 自顶向下
        - 自底向上
        - 由内向外
      - 设计方面
        - 实体与属性设计
          - 区分实体与属性
            - 原则：
              - 描述信息原则：实体有进一步的性质描述，属性无

        - 依赖性原则：属性单向依赖于实体存在
        - 一致性原则：一个实体有若干属性，属性有内在的关联性和一致性（例如学生的学号、姓名、年龄等在总体上协调一致，互相配合，构成一个完整的整体）
    - 实体与属性的详细描述
        - 实体与属性名：命名原则+名字特点+减少冲突+缩写原则
        - 确定实体标识：列出实体所有候选键，然后确定一个主键
        - 非空值原则：属性空值需要确定，主键不能有空值
- 联系与继承设计
    - 区分联系与继承
        - 语义区分
            - 继承：实体之间的分类与包含关系（同类事物）
                - 例如：学生-研究生、教师-教授-名誉教授
            - 联系：实体之间的一种广泛语义联系，反应实体内在的逻辑关联
                - 例如：教师指导学生、教师助手教师等
    - 联系的详细描述
        - 联系的种类
            - 存在性联系：学校有教师，教师有学生等
            - 功能性联系：教师授课、教师参与管理学生等
            - 事件联系：学生借书、学生打网球等
        - 与每个联系相关的实体集的个数
        - 联系的函数对应关系
        - 相同实体集间的多种联系
    - 继承的详细描述
        - 部分继承、全继承
        - 单继承、多继承
    - 案例二：篮球联赛管理
        - 实体有：球员、俱乐部、体育馆、比赛（比较隐晦，需要注意!）
    - 案例三：邮件系统管理





- 注意到这里面第二张图是用区间表示
  - 联系人可以发送 0~N 封邮件，而邮件只能由 1~1 个联系人发送；联系人可以接收 0~N 封邮件，但是邮件只能由 1~N 个联系人接收，不能没有联系人接收（要掌握这种表示方法）
- 案例四：旅游信息管理系统



例4 ER模型图

- 案例五：学生选课管理
  - 注意题目中没有明确提到的关系，即使现实中存在，也不要加上去。做题的话，就按照题目为准，嘿嘿。
- 视图集成

- 数据库的逻辑设计
  - 一对一
    - 均全参与：转换成一个关系模式
    - 一个全参与一个非全参与：转换成两个关系模式
    - 均非全参与：转换成三个关系模式
  - 一对多
    - 均全参与：转换成两个关系模式
    - 一个全参与一个非全参与：转换成三个关系模式
  - 多对多
    - 均非全参与：转换成三个关系模式
- 数据库的物理设计

数据库管理

- 数据库管理概述
  - 数据库的建立
  - 数据库的调整
  - 数据库的重组
  - 数据库的重构
  - 数据库的安全性控制与完整性识别
  - 数据库的并发控制
  - 数据库的故障恢复

- 数据库的监控
- 数据库管理的内容
  - 数据库的建立
    - 数据库运行环境的设置
    - 数据模式的建立
    - 会话环境的设置
    - 数据加载
  - 数据库的调整
    - 修改关系模式与视图使之更能适应用户的数据需求
    - 调整索引与集簇使数据库性能与效率更佳
    - 调整分区、调整数据库缓冲区大小以及调整并发度，是数据库物理性能更好
  - 数据库的重组
  - ……
- 数据库管理员 DBA
  - 有最高级别的权限，是管理数据库的核心人物（集合）
  - 需要谨慎设置权限，管理困难
- 数据库性能配置和优化
  - 数据库查询优化
    - 要会!! 合理选取查询命令对应的关系代数表达式，可以提高查询的效率
    - 关系代数的等价变换规则
      - ……有四个规则忘记记了
      - 选择的串接定律
      - 投影和选择的交换律（先投影再选择 F 等于先选择 F 再投影，注意 F 必须在选择的里面，不然的话会导致结果不一样）
      - 选择与笛卡尔乘积的交换律
      - 选择与自然连接的交换律
      - 选择与并运算的交换律
      - 选择与差运算的交换律
      - 投影与笛卡尔乘积的交换律
      - 投影与并运算的交换律
      - 笛卡尔乘积与连接运算的交换律
    - 查询优化策略与算法:
      - 选择运算尽可能先做->中间结果越少->磁盘 I/O 越少->越快
      - 可能需要先上移某个选择操作，然后将其下推到所有可能的分支（如使用视图的查询）
      - 同一对象的投影和选择晕眩应尽可能同时做(合并相邻的投影和选择运算的执行)
      - 将投影与其前后的二元运算结合起来做（合并投影和相邻的二元运算）
      - 下推投影
      - 将选择运算与其下面的笛卡尔乘积组合成连接运算
      - 消除重复结果元组的策略：消除 R 中的重复元组
      - 设计分组统计的策略：定义新操作，在允许的情况下引入新的投影和选择运算
    - 优化算法：见 PPT93 页

- 概念：InnoDB？ACID？