数据库复习

第一章、绪论

概念比较多

1.1数据库系统概述

数据库基本概念

> 1.数据：
>
> > 数据库中的基本对象，是描述事物的符号
>
> 2.数据库
>
> > 长期存贮在计算机中，有组织的，可共享的大量数据的集合
>
> 3.数据库管理系统（DBMS）
>
> > 用户与操作系统之间的一层软件，用于科学组织存储数据，高效获取和维护数据
> >
> > > sql server属于数据库管理系统
>
> 4.数据库系统（DBS）
>
> > 计算机系统引入数据库后的系统
> >
> > 构成：
> >
> > > DB
> > >
> > > DBMS
> > >
> > > Application System
> > >
> > > DataBase Administrator(DBA)

数据管理技术的产生与发展

> 1.人工管理阶段
>
> 2.文件系统阶段
>
> 3.数据库系统阶段

数据库系统的特点

> 1.整体数据结构化
>
> 2.数据的共享性高、冗余度低、易于扩充
>
> 3.数据独立性高

> 4.数据由DBMS同一管理和控制

1.2数据模型

分类

概念模型

逻辑模型和物理模型

组成要素

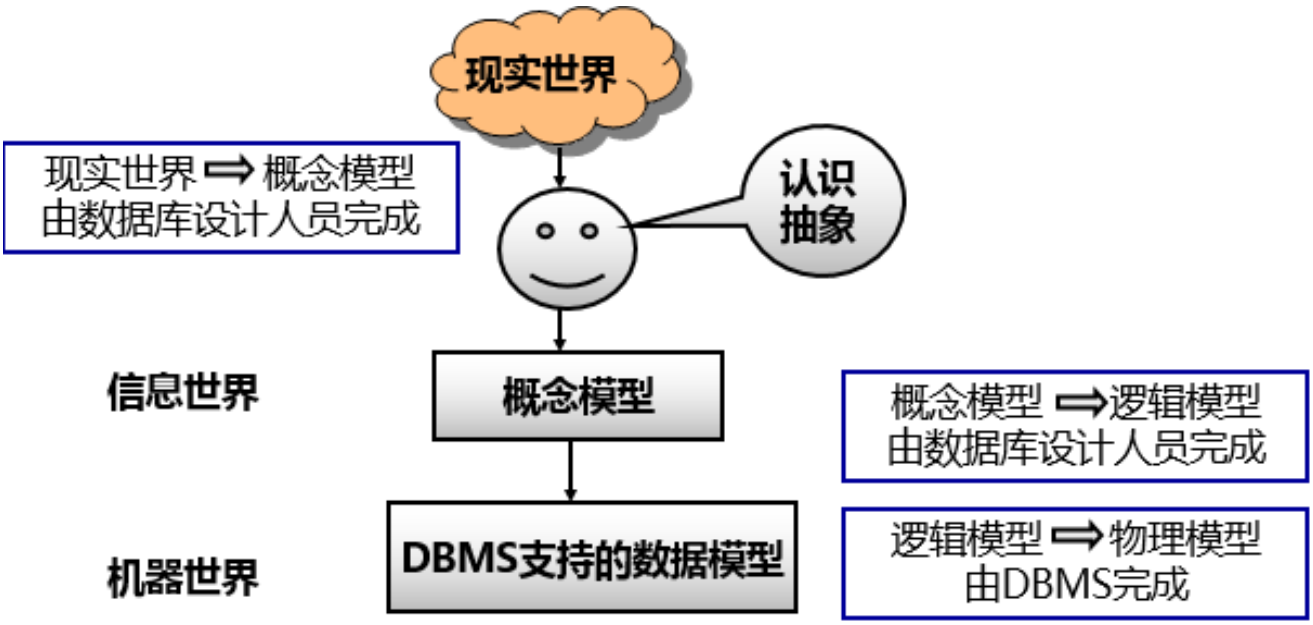
用来描述，组织，操作数据

> 1.数据结构【数据库组成对象及之间联系】
>
> 2.数据操作【查询、更新等】
>
> 3.完整性约束【制约依存规则】【保证数据的正确性、有效性、相容性】

概念模型和ER方法

概念模型又叫信息模型，强调语义表达能力，主要用于数据库设计,是 现实世界的第一层抽象

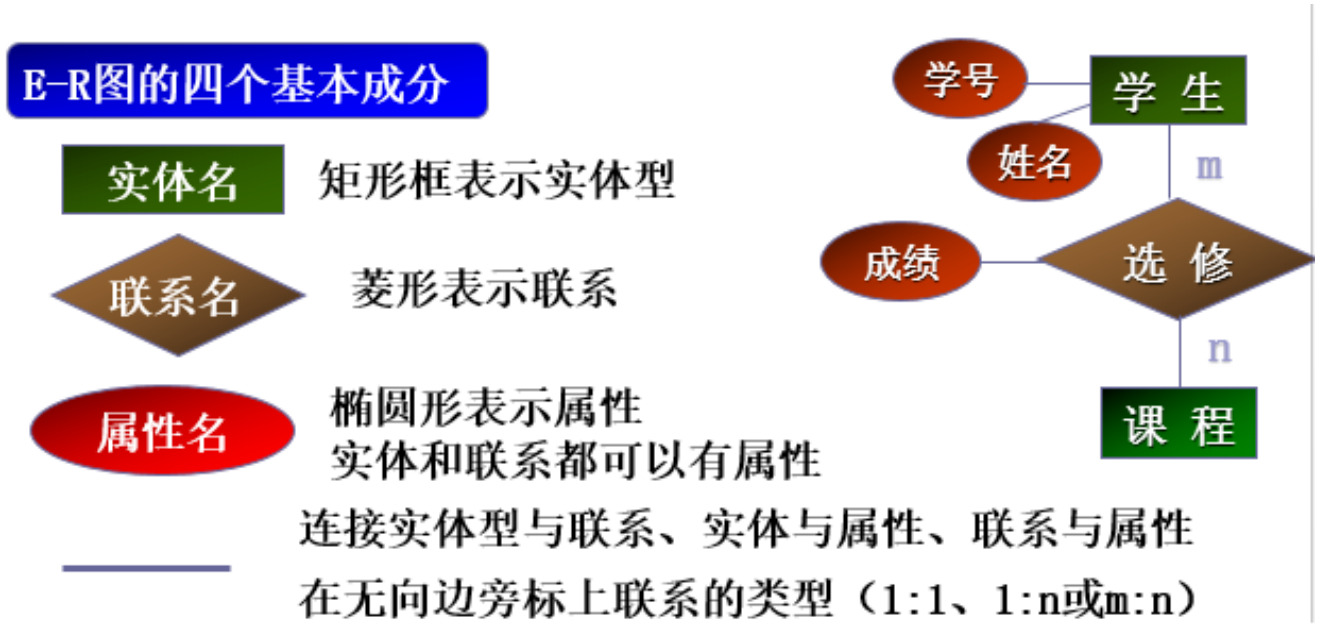
逻辑模型和物理模型按照计算机观点进行数据建模



> 概念模型：
>
> *实体
>
> *属性
>
> *码：唯一标识实体的属性集
>
> *域
>
> *实体型：用实体名和属性名集合来抽象和刻画同类实体
>
> *实体集：同一类实际集合

> *联系：分为一对一，一对多，多对多

ER模型



注意：

> 1.实体名、联系名不可以重复出现【不能由重名】
>
> 2.一个实体也不能多次出现【一个实体型就是一个矩形，不能画多了】

最常用的数据模型

> 1.层次模型【非关系模型】
>
> > 树图展示
> >
> > 优点：简单高效
> >
> > 缺点：不通用，插入删除复杂，访问复杂
>
> 2.网状模型【非关系模型】
>
> 3.关系模型
>
> 4.面向对象模型
>
> 5.对象关系模型

第二章、关系数据库

1.关系数据结构

> 候选码
>
> 主码
>
> 主属性：主码和候选码中的属性
>
> 非码属性

全码：关系中所有的属性都是候选码，称为主码

> 个人意见：斜体部分是“候选码”而不是“主属性”，这个是由条件的，自己想

外部码：外码。不是本关系的码，而是其他关系【可以是本关系，自己与自己链接，班长的那个例子】的码

外码不一定与对应的主码同名

关系模式：

$$< Empty\ Math\ Block >$$

$$R(U, D, DOM, F)$$

R:关系名

U:属性名集合

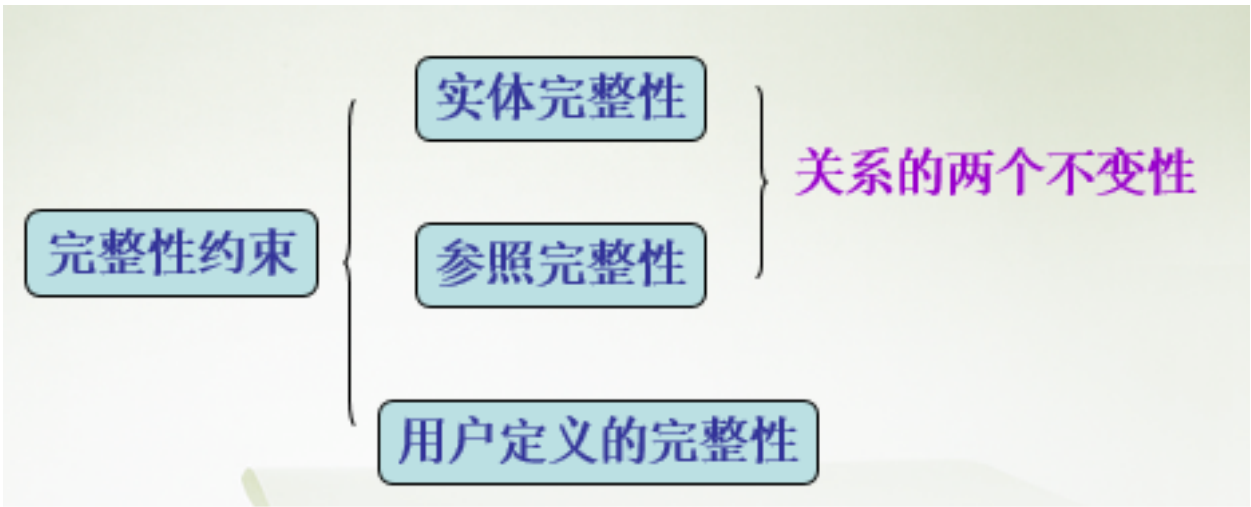
D：域集合

DOM：U到D的映射

F：U中属性之间的依赖集合

- 关系模式是型，是静态的、稳定的；
- 关系是关系模式的值，是动态的、随时间而变化的。

关系型数据库：



实体完整性：主属性不能为空

参照完整性：外码的取值

> 要么为空
>
> 要么等于对应的属性的主码

2.关系操作

集合运算$\cup \cap -$

就是集合运算那些东西

专门的关系运算$\sigma, \pi, \times, \div$

$$\sigma\Sigma\varsigma\tau T\upsilon\Upsilon\phi\Phi\varphi\chi X\psi\Psi\omega\Omega$$

tips： 在σ【选择】和π【投影】时，对应的列可以用数字代替，从左往右从1开始

笛卡儿积产生的结果中有大量的平庸元组需要剔除

连接

> 笛卡儿积 + 选择 = 条件连接

> 分类：
>
> 1.等值连接
>
> σ的条件是等号
>
> 2.自然连接
>
> 连接时的条件必须是相同的，连接后去除重复属性列
>
> 3.外连接
>
> 左外连接
>
> 保留左侧要舍弃的元组
>
> 右外联结
>
> 保留右侧要舍弃的元组

÷

> 象集：Z_x1在关系中取$x = x_1$的元组，并去除x列
>
> 除法：n给定关系R (X, Y) 和S(Y, Z)，X、Y、Z为属性组。¨元组在X上分量值x的象集Yx包含S在Y上投影的集合。

3.关系完整性约束

4.关系代数运算

第三章、关系数据库标准语言SQL

1.SQL特点

！@#%！@%#￥@……@#￥%&……%#￥&%￥@￥！#￥！%！#￥……！#￥@#！%

2.SQL使用——数据定义

模式

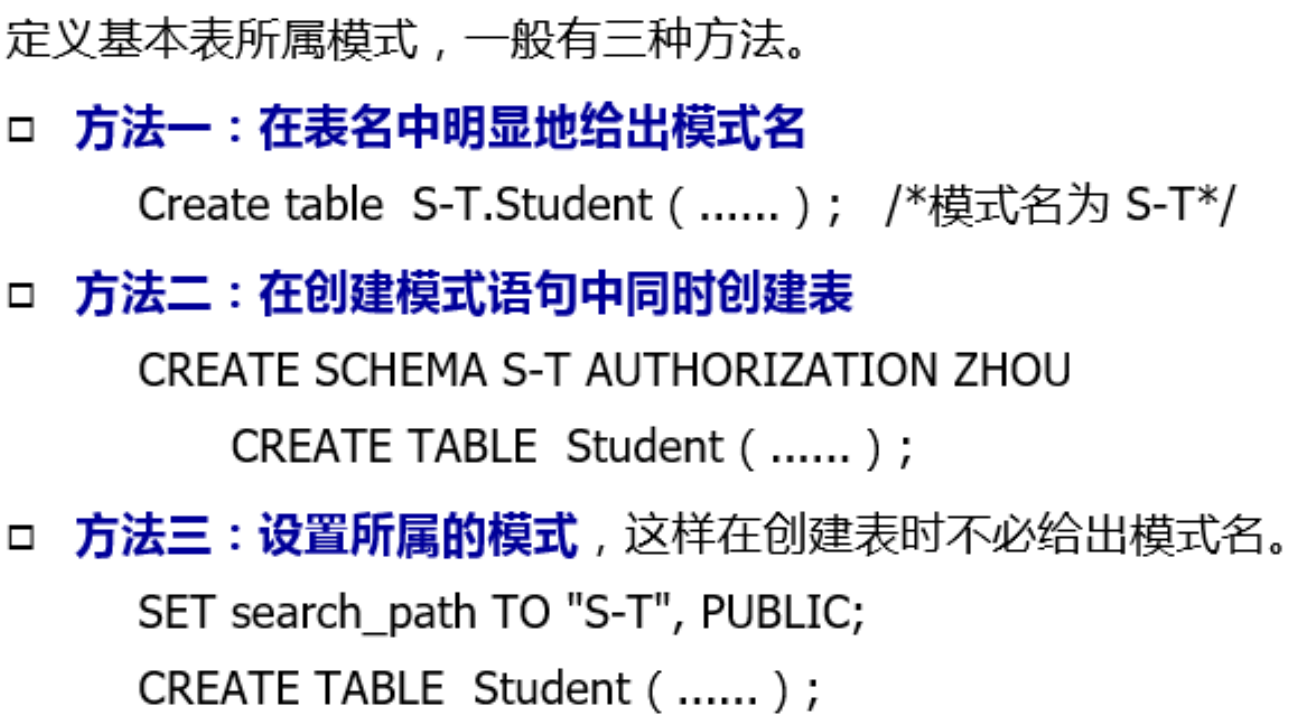
```
CREATE SCHEMA "模式名" AUTHORIZATION “用户名”

DROP SCHEMA "模式名" [CASCADE | RESTRICT]
```

基本表

表的建立

```
CREATE TABLE SC
  (      Sno   CHAR(9),
         Cno   CHAR(4),
         Grade     SMALLINT,
         PRIMARY KEY (Sno, Cno), /* 主码由两个属性构成，  必须作为表级完整性进行定义*/
         FOREIGN KEY (Sno) REFERENCES Student(Sno),
          /* 表级完整性约束条件，Sno是外码，被参照表是Student */
         FOREIGN KEY (Cno) REFERENCES Course(Cno)
          /* 表级完整性约束条件，  Cno是外码，被参照表是Course*/
  );
```



表的修改

```
ALTER TABLE <表名>
              [RENAME <旧表名> TO <新表名>];
            [ADD <新列名> <数据类型>[完整性约束]]
            [DROP <列名> ]
             [ALTER COLUMN <列名> <数据类型>]
            [ADD <完整性约束>]
            [DROP <完整性约束名>]
```

表的删除

```
DROP  TABLE  <表名> ［RESTRICT| CASCADE］;
```

索引

索引分类

> 唯一索引：
>
> > 要求对应的列没有重复值
>
> 聚簇索引：
>
> > 改变对应的物理顺序
> >
> > 一个表只能有一个聚簇索引

索引建立与删除

```
CREATE  [UNIQUE] [CLUSTER]  INDEX <索引名>
        ON <表名> (<列名>[<次序ASC|DESC>],[, <列名>[<次序>]]…);
DROP INDEX <表名.索引名>;
```

视图

见视图专题

3.SQL使用——数据查询

格式

```
 SELECT [ALL|DISTINCT] <目标列表达式>
                      [, <目标列表达式>]……
    FROM  <表名或视图名>[,<表名或视图名>] …
   [WHERE  <条件表达式>]
   [GROUP  BY <列名1> [HAVING  <条件表达式>]]
   [ORDER BY <列名2>[ASC|DESC]];
```

查询分类

> 单表查询
>
> 多表查询
>
> > 注意：外连接`left out join on(关系)`,`right out join on (关系)`,`out join(关系)`
>
> 嵌套查询
>
> > 分类：
> >
> > 相关子查询
> >
> > 不相关子查询
> >
> > 聚集函数
> >
> > ANY,ALL

> IN、EXISTS 使用NOT EXISTS取反实现全称量词】
>
> 集合查询【UNION INTERSECT EXCEPT】
>
> > 注意：
> >
> > > 1.要排序的话只能在最后排一次
> > >
> > > 2.操作的两部分对应的列要完全一致

4.SQL使用——数据更新

插入

```
INSERT INTO <表名>[(<列名1>[,<列名2>]…)]
     VALUES( <常量1>[,<常量2>]…);


INSERT  INTO   Dept_age( Sdept, Avgage )
        SELECT Sdept, AVG(Sage)
        FROM   Student
        GROUP BY Sdept;
```

修改

```
   UPDATE <表名>
   SET  <列名>=<表达式> [,<列名>=<表达式>]…
```

删除

```
 DELETE
 FROM    <表名>
 [WHERE  <条件>];
```

5.SQL使用——视图专题

视图创建

```
CREATE VIEW <视图名>[(<列名>[,<列名>]…)]
      AS 子查询
      [WITH CHECK OPTION];
```

视图删除

```
DROP  VIEW  <视图名> [ CASCADE ] ;
```

视图消解【重要】

出错原因：消解是机械式的，可能会把聚集函数整到WHERE里面，造成错误

视图更新

不可更新的视图：【了解】

> •由多个表导出的视图，不可更新
>
> •视图的列来自表达式或常数，不可插入、修改、可删除
>
> •视图列是来自集函数，不可更新
>
> •视图定义中含有GROUP BY子句，不可更新
>
> •视图定义中含有DISTINCT短语，不可更新
>
> •视图定义中内层嵌套的表与导出该视图表相同，不可更新
>
> •在不允许更新的视图上定义的视图，不可更新

第四章、数据库安全性

用户识别

> 用户标识
>
> 口令

存取控制

数据库安全最重要的一点就是确保只授权给有资格的用户访问数据库的权限

方法分类

> 自主存取控制（DAC）

```
GRANT <权限> [, <权限> ]……
      [ ON <对象类型><对象名>]
      TO <用户>[, <用户>]……
      [ WITH GRANT OPTION ];



REVOKE <权限> [, <权限> ]……
      [ ON <对象类型><对象名>]
      FROM <用户>[, <用户>]……
```

不同对象类型允许的操作权限

对 象	对象类型	操作权限
属性列	TABLE	SELECT, INSERT, UPDATE, DELETE, ALL PRIVILEGES
视 图	TABLE	SELECT, INSERT, UPDATE, DELETE, ALL PRIVILEGES
基本表	TABLE	SELECT, INSERT, UPDATE, DELETE, ALTER, INDEX, ALL PRIVILEGES
数据库	DATABASE	CREATETAB

> 强制存取控制(MAC)
>
> > 敏感度标记（Label）。敏感度标记被分为若干级别，例如：
> >
> > 绝密（TopSecret）
> >
> > 机密（Secret）
> >
> > 可信（Confidential）
> >
> > 公开（Public） 等
> >
> > ¨主体的敏感度标记称为许可证级别（ClearanceLevel）；
> >
> > ¨客体的敏感度标记称为密级（Classification Level）。

DAC与MAC共同构成DBMS的安全机制

¨实现MAC时要首先实现DAC，原因是：较高安全性级别提供的安全保护要包含较低级别的所有保护。

第五章、数据库完整性

分类：

> 实体完整性
>
> > primary key
>
> 参照完整性

被参照表（例如Student）	参照表（例如SC）	违约处理
可能破坏参照完整性	← 插入元组	拒绝
可能破坏参照完整性	← 修改外码值	拒绝

> 用户自定义完整性,限制的主要是参照表

用户自定义完整性

> UNIQUE 、NOT NULL 、CHECK
>
> > CHECK的写法：就像sql查询中where条件里那么写就行了

触发器

```
CREATE  TRIGGER  <触发器名>
             { BEFORE | AFTER }  <触发事件> ON <表名>
              FOR  EACH  {  ROW | STATEMENT  }
              [ WHEN <触发事件>]
              <触发动作体>
//SQL SERVER 以这个为主吧
CREATE  TRIGGER   触发器名
       ON  数据表名或视图名
       AFTER | FOR| Instead of   INSERT 或 DELETE 或 UPDATE
AS
BEGIN
        --这里是要运行的SQL语句
END
GO
```

deleted表和inserted表

deleted 表和 inserted 表——临时表

- 用于存放对表中数据行的修改信息，在触发器执行时自动创建的，当触发器工作完成，被删除。
- Deleted 表用于存储 DELETE 和 UPDATE 语句所影响的行的副本。
- Inserted 表用于存储 INSERT 和 UPDATE 语句所影响的行的副本。
- 插入操作（Insert）
  Inserted表有数据，Deleted表无数据。
- 删除操作（Delete）
  Inserted表无数据，Deleted表有数据。
- 更新操作（Update）
  Inserted表存有新数据，Deleted表存有旧数据。

第六章、关系数据理论

引入：关系模式在前边就已经提到过了，这里更多的讨论的是属性见关系，所以将关系模式表示为关系模式名R，属性名U,属性间关系F的三元组$R(U,F)$

数据依赖的分类

1.函数依赖【我们后边讨论的都是这个】

2.多值依赖

函数依赖的分类

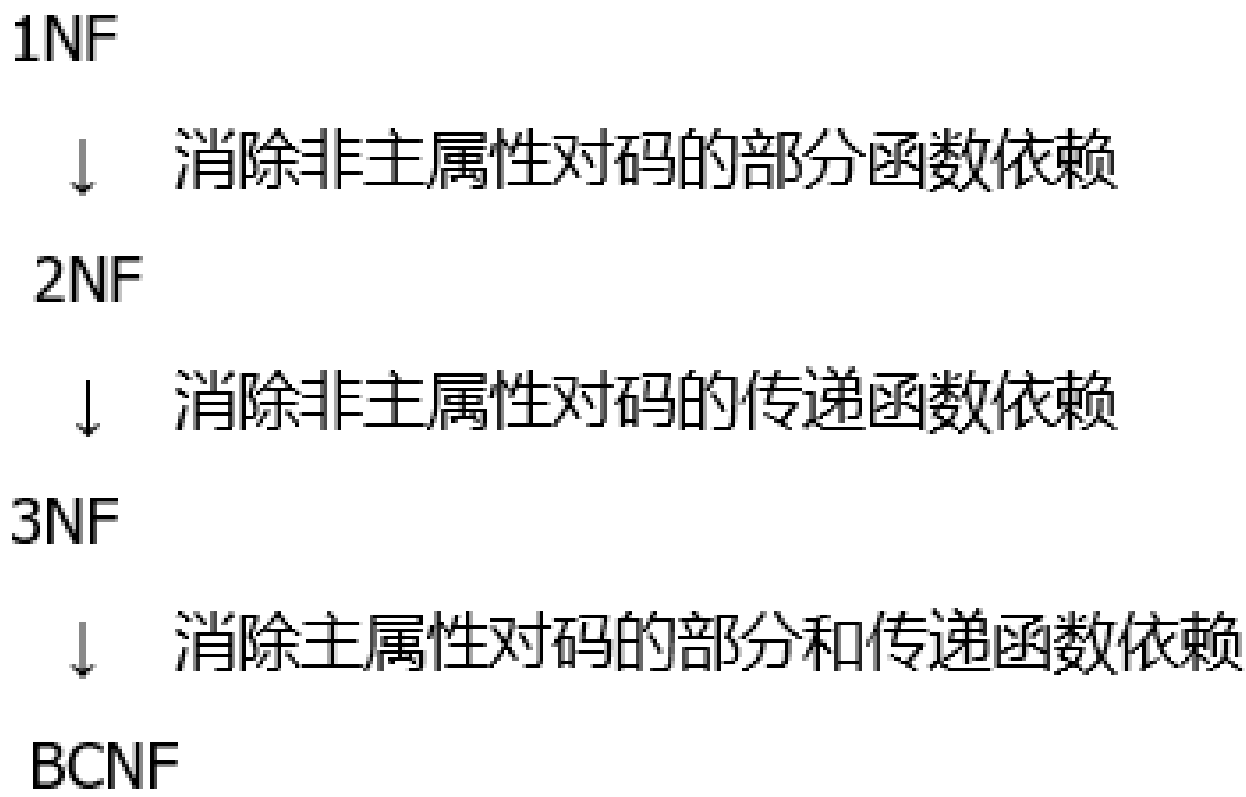
> 平凡函数依赖：（类似$AB \to A$）
>
> 非平凡函数依赖：(类似$AB \to C$)

> 完全函数依赖：（类似：$AB \to C$，但是$A \nrightarrow C$而且$B \nrightarrow C$）
>
> 部分函数依赖：(类似：$AB \to C$,但是存在一个现象$A \to C$)

> 直接函数依赖：
>
> 传递函数依赖：($A \to B, B \to C \Longrightarrow A \to C$)

范式

范式分类：



Armstrong公理系统

定位：一套用于模式分解算法的推理规则

自反律

增广律

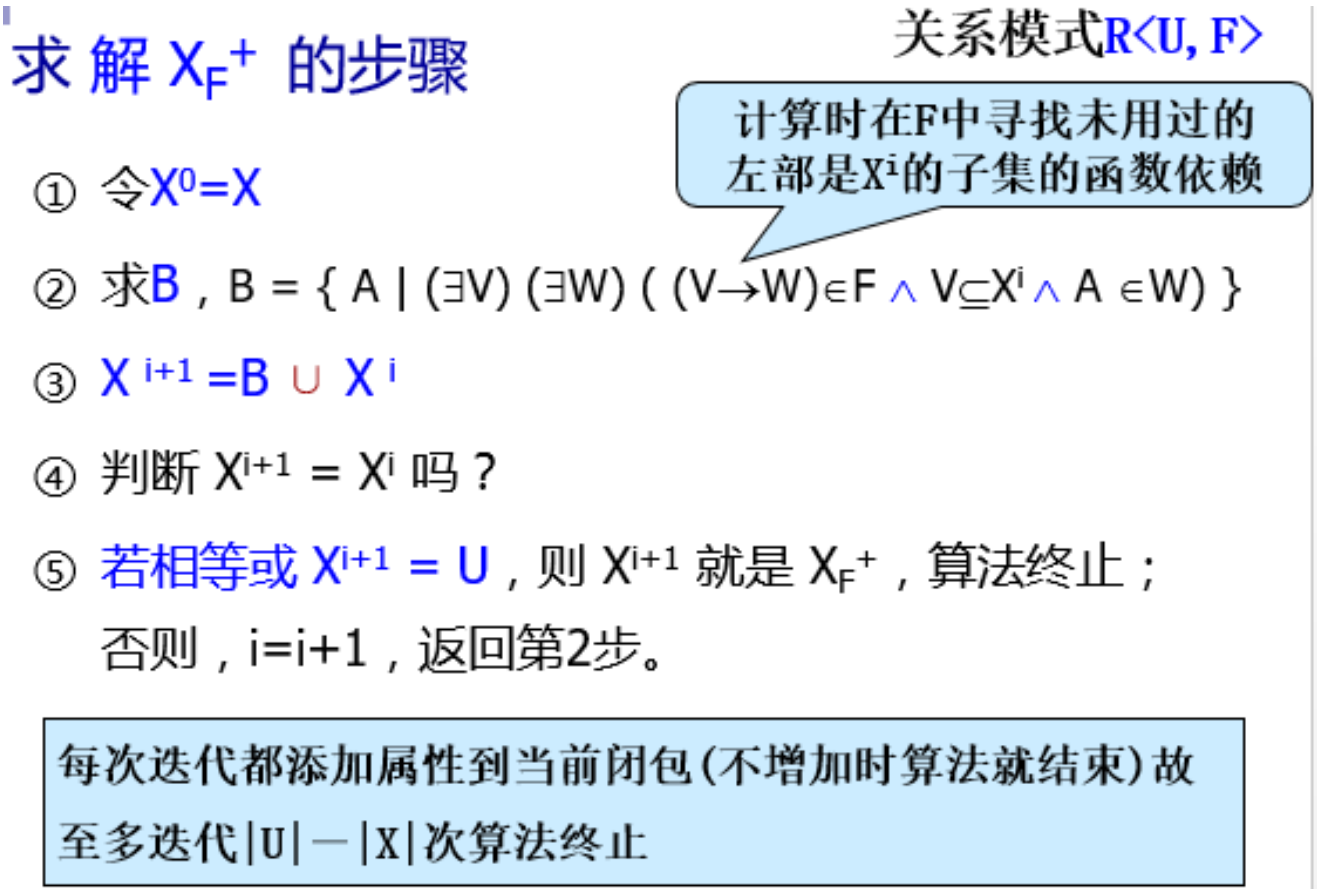
传递律

合并规则：

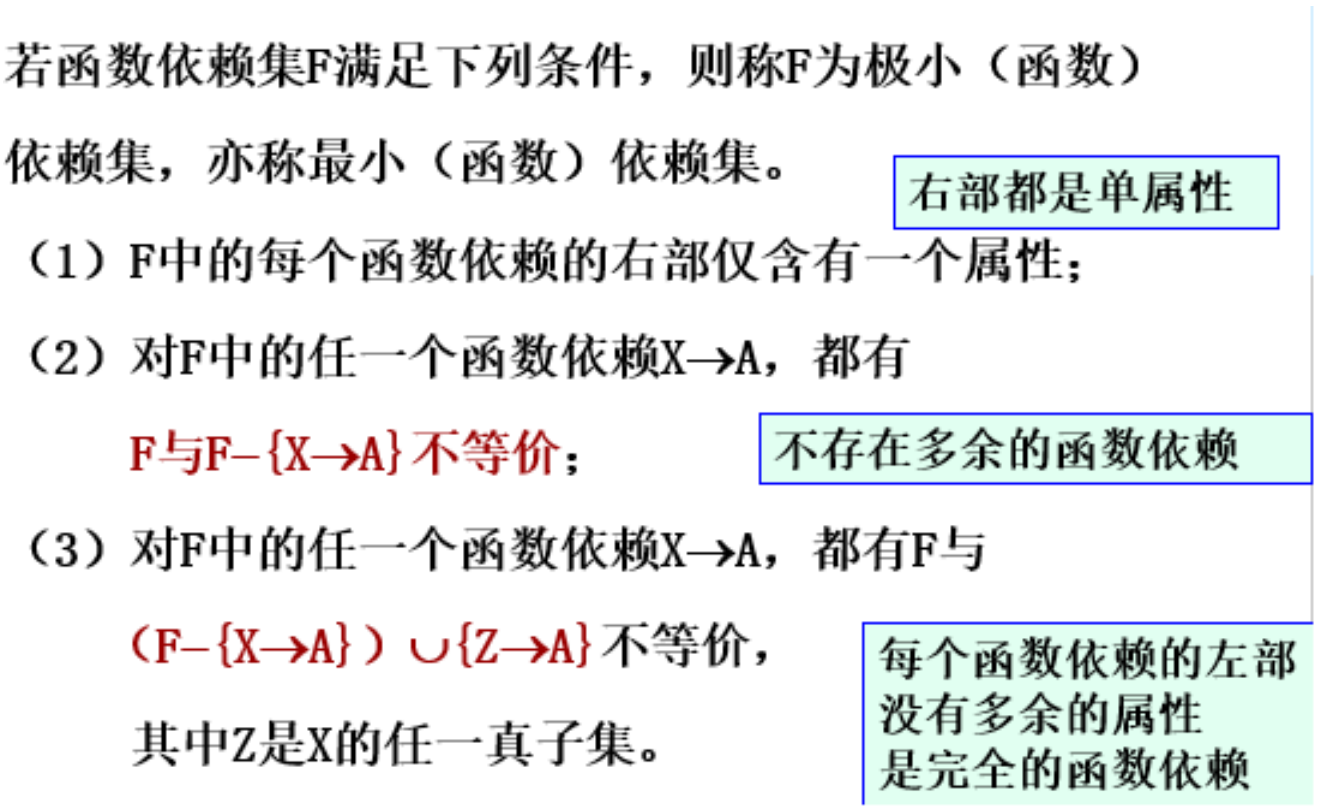
分解规则：

伪传递规则：$X \to Y, YZ \to A \Longrightarrow XZ \to A$

闭包



最小依赖集



求最小依赖集的步骤：

> 1.对右部的属性进行分解，保持右部只有一个属性

> 2.依次检查每一个公式是够必须的公式
>
> > 去掉此公式，然后求左侧属性的闭包，看能不能包括右侧属性
>
> 3.依次检查每个公式左侧的每一个属性是否此公式必须的属性
>
> > 一个一个的尝试去掉左侧的属性，【但是保留本公式】，看剩下的属性的闭包能否包含右侧的属性

求解关系模式的候选码

属性分类：

- L类：只出现在函数依赖的左边的属性
- R类：只出现在函数依赖的右边的属性
- N类：在函数依赖的两边均未出现的属性
- LR类：出现在函数依赖的两边的属性

模式分解

要求：保证分解后的关系模式与原关系模式等价【。。。。】

等价包括：

> 分解具有“无损连接性”（Losslessjoin）
>
> 分解要“保持函数依赖”（Preservedependency）
>
> 分解既要“保持函数依赖”，又要具有“无损连接性”

分解为第三范式：

保留函数依赖：

> F最小化
>
> 一个推导式弄做一个模式

保留函数依赖和无损联结：在上边的基础上加一个关系模式，里边由候选码关键字【所有的主属性】组成

分解为BCNF

- 按自顶向下的方式，从关系模式R<U，F>开始，选择不是BCNF的模式将其一分为二，直到都为BCNF。（二叉分解）
- 求R的候选码

```
R的分解={R};
while(R的分解中有一项S不是BCNF)
{  在S中找出X→A，X不是S的码;
   S1=XA;
   S2=S-A;
   用{S1,S2}代替S;
}
```

第七章、数据库设计

设计的是逻辑结构和物理结构

基本步骤

> 1.系统需求分析阶段
>
> > 使用自顶向下的结构化分析方法(SA)，逐层分析
> >
> > > SA方法——数据流图
> > >
> > > SA方法——数据字典（数据项、数据结构、数据流、数据存储、处理过程）
>
> 2.概念结构设计阶段【重点】
>
> > 将第一步的抽取为信息结构模型【概念模型】【ER图】（ER图的东西去找第一章的去）
>
> 3.逻辑结构设计阶段【重点】
>
> > 1.一个实体型转换为一个关系模式
> >
> > 2.一个m:n联系转换为一个关系模式
> >
> > 3.1:n联系
> >
> > > ①转换为一个独立的关系模式
> > >
> > > ②与n端对应的关系模式合并
> >
> > 4.1:1联系
> >
> > > ①转换为一个独立的关系模式

> > ②与某一端对应的关系模式合并
> >
> > 5.三个以上实体间的多元联系转换为一个关系模式
> >
> > 6.同一实体集的实体间的联系，即自联系，也可按上述1:1、1:n和m:n三种情况分别处理
> >
> > 7.具有相同码的关系模式可合并
>
> 4.物理设计阶段
>
> 5.数据库实施阶段
>
> 6.数据库运行与维护阶段

第九章、关系系统和查询优化

关系系统

关系系统就是之前讲的关系数据库的结构、操作、完整性什么的了

关系系统的分类

> 表式系统
>
> （最小）关系系统
>
> 关系完备的系统
>
> 全关系系统

关系系统的查询优化

查询处理的步骤：查询分析$\longrightarrow$查询检查$\longrightarrow$查询优化$\longrightarrow$查询执行

能根据假设的情景算一下查询调入调出多少磁盘模块

优化的原则：就那些字面上的意思

> 查询优化的一般准则
>
> 1.选择运算应尽可能先做（目的：减小中间关系）
>
> 2.在执行连接操作前对关系适当进行预处理：按连接属性排序；在连接属性上建立索引
>
> 3.投影运算和选择运算同时做（目的：避免重复扫描关系）
>
> 4.将投影运算与其前面或后面的双目运算结合
>
> （目的：减少扫描关系的次数）
>
> 5.某些选择运算＋在其前面执行的笛卡尔积 ===> 连接运算
>
> 例如：σS.Sno=SC.Sno(S╳SC) ===> S⋈SC
>
> 6.提取公共子表达式

代数优化

优化的步骤

> “李氏”步骤：
>
> 1.写出表连接的骨架和最后选择的属性
>
> 2.从树的上边往下添加选择
>
> 3.从树的上边往下添加投影
>
> 4.检查是不是有的选择、投影是多的

最后（如果可以的话）笛卡儿积和前边的选择会转化为自然连接

第十章、数据库恢复技术

事务

```
BEGIN TRANSACTION
    SQL 语句1
    SQL 语句2
    ……
COMMIT 或 ROLLBACK
```

COMMIT:提交【成功】

ROLLBACK:回滚【失败】

事务特性

> 原子性
>
> > 事务不能分割
>
> 一致性
>
> > 数据库中只有事务成功执行后的结果
>
> 隔离性
>
> > 对数据的访问是临界的
>
> 持续性
>
> > 事务成功执行后对数据库的操作是永久的

故障的种类

> 1.事务故障
>
> 撤销事务，强行回滚
>
> 2.系统故障
>
> 3.介质故障【破坏性很大】
>
> 4.计算机病毒

恢复技术

实现原理：冗余

建立冗余方式：数据转储，日志文件

数据转储

		转储状态	
		动态转储	静态转储
转储方式	完全转储	动态完全转储	静态完全转储
	增量转储	动态增量转储	静态增量转储

日志文件

分类：以记录为单位，以数据块为单位

内容：开始标记，结束标记，更新操作

先更新日志文件，后更新数据库

恢复策略

事务故障：UNDO

系统故障：

系统故障的恢复步骤

1. 正向扫描日志文件（即从头扫描日志文件）
   - Redo队列: 在故障发生前已经提交的事务
   - Undo队列:故障发生时尚未完成的事务
2. 反向扫描日志文件，对每个UNDO事务的更新操作执行逆操作。
3. 正向扫描日志文件，对每个REDO事务重新执行登记的操作。

先REDO，再UNDO

介质故障：

1.装入最新副本

> 静态转储直接ok
>
> 动态转储导入数据库副本和项目日志，使用数据库故障恢复方法恢复

2.装入相关的日志文件副本，REDO

具有检查点的恢复技术

检查点：统计备份时未提交的事务目录，这样以后用这个备份时就不用检索日志获得UNDO队列了

利用检查点的恢复步骤

1. 从重新开始文件中找到最后一个检查点记录在日志文件中的地址，由该地址在日志文件中找到最后一个检查点记录
2. 由该检查点记录得到检查点建立时刻所有正在执行的事务清单ACTIVE-LIST，建立两个事务队列：UNDO-LIST 和REDO-LIST 。把ACTIVE-LIST暂时放入UNDO-LIST队列，REDO队列暂为空。

3. 从检查点开始正向扫描日志文件，直到日志文件结束。
    - 如有新开始的事务Ti，把Ti暂时放入UNDO-LIST队列
    - 如有提交的事务Tj，把Tj从UNDO-LIST队列移到REDO-LIST队列
4. 对UNDO-LIST中的每个事务执行UNDO操作，对REDO-LIST中的每个事务执行REDO操作。

数据库镜像。。。。

第十一章、并发控制

【应该不是重点吧】

并发操作带来的不一致性

> 丢失修改
>
> > 两个事务并发修改，先改的会被覆盖
>
> 不可重复读
>
> > 读数据后数据被更新
>
> 读脏数据
>
> > 更新了读，读了后更新数据的事务被撤销

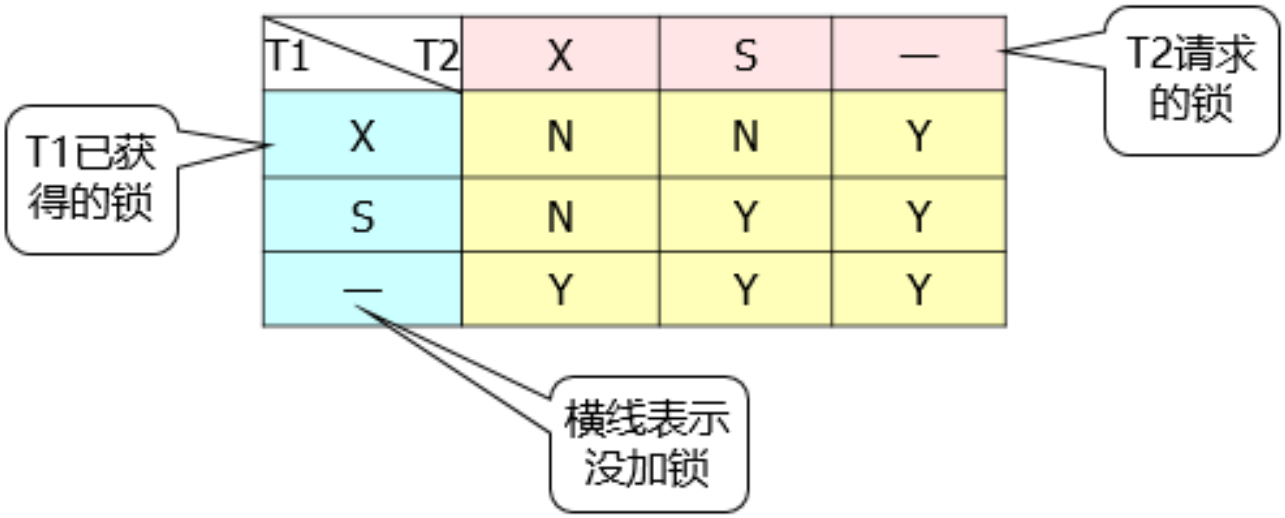
锁

写锁：X

读锁：S

基本锁的相容矩阵

- 最左边一列表示T1已经获得数据对象上的锁的类型。
- 第一行表示T2对同一数据对象发出封锁请求。
- Y表示T2的封锁要求与T1已经持有的锁相容，封锁可以满足。
- N表示T2的封锁请求与T1已经持有的锁冲突，T2的请求被拒绝。



活锁和死锁

活锁：不断有人插队⟶使用先来先服务

死锁：连个环⟶

预防死锁

> 一次封锁法
>
> 顺序封锁法

诊断和解除

> 超时法
>
> 等待图法【合理】

并发调度的可串行性

如果并发的结果和某种顺序的串行结果一致，则并发时没有出现错误的

两段锁协议

一致申⟶一致放

分为：扩展阶段和收缩阶段

封锁粒度

封锁的对象大小

粒度↑⟶锁的数量↑，系统开销↑，并发程度↑

n在多粒度封锁中一个数据对象可能以两种方式封锁：

显式封锁: 直接加到数据对象上的封锁

隐式封锁: 由于其上级结点加锁而使该数据对象加上了锁