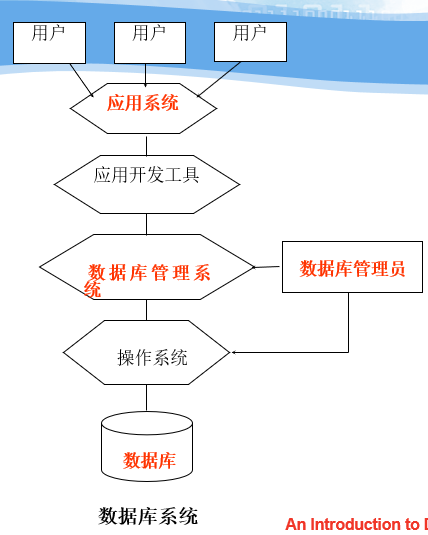
1.2.3本文中为一、(一)、1.下面再有就是(1) a. a.1 a.1.1

包含于⊆ 属于∈ 蕴含→ 存在$ 任意# 非~

数据库技术产生于六十年代末，是数据管理的最新技术，是计算机科学的重要分支。数据库技术是信息系统的核心和基础，它的出现极大地促进了计算机应用向各行各业的渗透。数据库的建设规模、数据库级信息量的大小和使用频度已成为衡量一个国家信息化程度的重要标志。

一、绪论

(一)数据库系统概述

1.四个基本概念：数据、数据库、数据库管理系统、数据库系统

(1)数据

数据的定义：数据是数据库中存储的基本对象

数据的定义：描述食物的符号记录

数据的种类：文本、图形、图像、音频、视频、学生的档案记录、货物的运输情况

数据的特点：数据与其语义是不可分的，数据的含义称为语义

(2)数据库

数据库的定义：数据库是长期存储在计算机内、有组织的、可共享的大量数据的集合

数据库的基本特征：数据按一定的数据模型左值、描述和存储；可为各种用户共享；冗余度最小；数据独立性高；容易扩展

(3)数据库管理系统(Database Management System)

DBMS的定义：位于用户和操作系统之间的一层数据管理软件；是基础软件、是一个大型复杂的软件系统

DBMS用途：科学的组织和存储数据、高效地获取和维护数据

主要功能：

数据定义功能：提供数据定义语言(DDL)；定义数据库中的数据对象；数据组织、存储、管理：分类组织、存储和管理各种数据；确定组织数据的文件结构和存取方式；实现数据之间的联系；提供多种存取方法提高存取效率

数据操纵功能：提供数据操纵语言(DML)实现对数据可的基本操作(查询、插入、删除、修改)

数据库的事务管理和运行管理：提供事务控制语言(TCL)；数据库在建立、运行和维护时由DBMS统一管理和控制，保证数据的安全性、完整性、多用户对数据的并发使用发生故障后的系统恢复

数据库的建立和维护功能(使用程序)：数据库初始数据装在转换；数据库转储；价值故障恢复；数据库的重组织；性能监视分析等

其他功能：DBMS于网络中其他软件系统的通信；两个DBMS系统之间的数据转换；异构数据库之间的互访和互操作

数据库控制语言(DCL)

(4)数据库系统

数据库系统定义：在计算机系统中引入数据库之后的系统构成

数据库系统的构成：数据库；数据库管理系统；应用程序；数据库管理员(数据库管理员DBA、系统分析员、数据库设计人员、应用程序员、用户)

e.g:

某大公司工资系统。指出那些人员可以执行下面的功能：

1)编写一个应用程序以生成和打印帐单(程序员)

2)改变一个已经搬迁的员工在数据库中的地址信息(数据库管理员)

3)为新来的职工建立一个新的用户帐号(数据库管理员)

2.数据管理技术的产生和发展

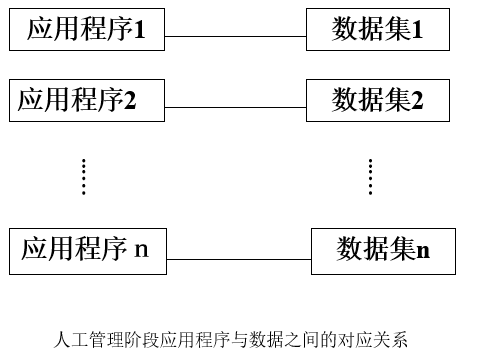
数据管理：对数据进行分类、组织、编码、存储、检索和维护；数据处理的中心问题

数据管理技术的发展过程：

人工管理阶段(20世纪40年代中--50年代中)

文件系统阶段(20世纪50年代末--60年代中)

数据库系统阶段(20世纪60年代末--现在)

数据管理技术的发展动力：应用需求的推动；计算机硬件的发展；计算机软件的发展

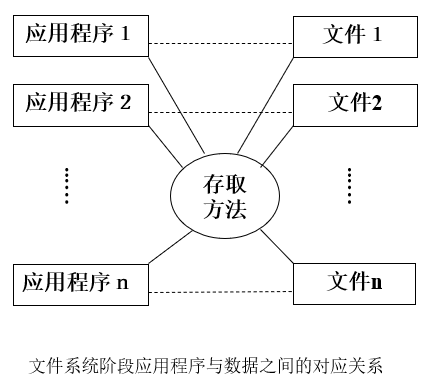
(三个阶段中的特点都是针对数据的特点)

(1)人工管理阶段：

时间：20世纪40年代中--50年代中

产生背景：应用需求 --> 科学计算；硬件水平-->无直接存取存储设备；软件水平-->没有操作系统；处理方式-->批处理；

特点：管理者-->用户(程序员)，数据不保存；面向的对象-->某一应用程序；共享程度-->无共享；独立性-->不独立，完全依赖程序；结构化-->无结构; 控制能力-->应用程序自己控制



(2)文件系统阶段

时间：20世纪50年代末--60年代中

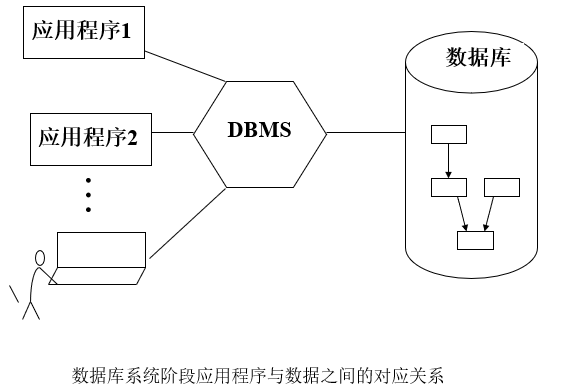
产生背景：应用需求-->科学计算、管理；硬件水平-->磁盘、磁鼓；软件水平-->有文件系统；处理方式-->联机实时处理、批处理

特点：管理者-->文件系统，数据可长期保存；面向的对象-->某一应用(课件上是应用程序)；共享程度-->共享性差，冗余度大；结构化-->记录内有结构，整体无结构；独立性-->独立性差，数据的逻辑结构改变必须修改应用程序；控制能力-->应用程序自己控制

文件系统中数据的结构：记录内有结构；数据的结构是靠程序定义和解释的；数据只能是定长的-->可以间接实现数据变长要求，但访问相应数据的应用程序复杂了；文件之间是独立的，因此数据整体无结构-->可以间接时间数据整体的有结构，但必须在应用程序中描述对数据间的联系；数据的最小存取单位是记录(比较大)

(3)数据库系统阶段

时间：20世纪60年代末--现在



产生背景：应用背景-->大规模管理；硬件背景-->大容量磁盘、磁盘阵列；软件背景-->有数据库管理系统；处理方式-->联机实时处理，分布处理，批处理

3.数据库系统的特点

(1)特点

管理者-->数据库管理系统；面向的对象-->现实世界(一个部门、企业、组织等)；共享程度-->共享性高，冗余度小，易扩充；独立性-->具有高度的物理独立性和一定的逻辑独立性；结构化-->整体结构化，用数据模型描述；控制能力-->由数据库管理系统统一提供数据安全、完整性、并发控制和恢复能力

(2)数据结构化

整体数据的结构化是数据库的主要特征之一-->不在仅仅针对某一个应用，而是面向全组织；不仅数据内部结构化，整体是结构化的，数据之间具有联系

数据库中实现的是数据的真正结构化-->数据的结构用数据模型描述，无需程序定义和解释；数据可以变长；数据的最小存取单位是数据项

(3)数据的共享性高，冗余度低，易扩充

数据库系统从整体角度看待和描述数据，数据面向整个系统，可以被多个用户、应用共享使用

数据共享的好处；较少数据冗余，节约存储空间；避免数据之间的不相容性和不一致性；使系统易于扩充

(4)数据独立性高

物理独立性：用户的应用程序与存储在磁盘上的数据库中数据是相互独立的，当数据的物理存储改变了，应用程序不用改变

逻辑独立性：用户的应用程序与数据库的逻辑结构是相互独立的数据的逻辑结构改变了，用户程序也可以不变

数据独立性是由DBMS的二级映像功能来保证的

(5)数据由DBMS统一管理和控制

数据的安全性保护：保护数据一档只不合法的使用噪声的数据的泄密和破坏

数据的完整性检查：将数据控制在有效的范围内，或者保证数据之间满足一定的关系

并发控制：对多用户并发操作加以控制和协调，防止相互干扰而得到错误的结果

数据库恢复：将数据库从错误的状态恢复到某一已知的正确状态

综上所述，数据库是长期存储在计算机内有组织、大量、共享的数据集合。他可以汞各种用户共享，具有最小冗余度和较高的数据独立性，数据库管理系统在数据库建立、运用和维护时对数据库进行统一控制，以保证数据的完整性和安全性，并在多用户同时使用数据可是进行并发控制，在发生故障后对数据库进行恢复

数据库系统的出现使信息系统从以加工数据的程序为中心转向围绕共享的数据库为中心的新阶段

(二)数据模型

数据模型也是一种模型，是对现实世界数据特征的抽象。即数据模型是用来描述数据、组织数据和对数据进行操作的。数据模型是数据库的核心和基础

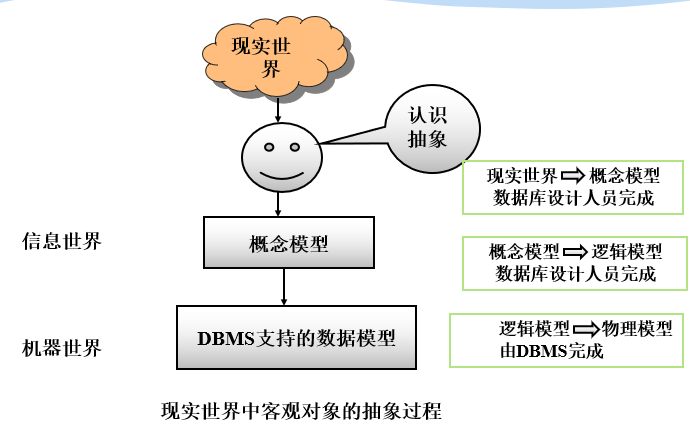
概念引入

现实世界：现实世界即客观存在的世界，各种事物及事物之间的联系。一个事物可以有许多特征，通常都是选用人们感兴趣的以及最能表征该事物的若干特征来描述该事物。以人为例，常选用姓名、性别、年龄、籍贯等描述一个人的特征。事物间的关联是多方面的。

信息世界：现实世界中的事物及其联系由人们的感官感知，经过人们头脑的分析、归纳、抽象，形成信息。对这些信息进行记录、整理、归类和格式化后，它们就构成了信息世界。对所研究的信息世界建立一个抽象的模型，称之为信息模型(即概念模型)。目前较为流行的一种信息模型是实体联系模型。

机器世界：用计算机管理信息，必须对信息进行数据化，数据化后的信息称之为数据，数据是能够被机器识别并处理的。数据化了的信息世界称之为机器世界。

在数据库中用数据模型这个工具来抽象、表示和处理现实世界中的数据和信息。通俗地讲数据模型就是现实世界的模拟。数据模型应满足三方面要求：能比较真实地模拟现实世界；容易为人所理解；便于在计算机上实现

1.两大类数据模型

数据模型分为两类(分属两个不同的层次)

概念模型：也称信息模型，按用户的观点对数据和信息建模，用于数据库设计

概念模型将显示世界抽象为信息世界

逻辑模型和物理模型：逻辑模型主要包括网状模型、层次模型、关系模型、面向对象模型等，按计算机系统的观点对数据建模，用于DBMS实现

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法和存取方法，在磁盘或者磁带上的存储方式和存取方法

客观对象的抽象过程-->两步抽象：将现实世界中的客观对象抽象为概念模型；把概念模型转换为某一DBMS支持的数据模型

2.数据模型的组成要素：数据结构；数据操作；完整性约束条件

(1)数据结构

定义：描述数据库的组成对象以及对象之间的联系

描述的内容：与数据类型、内容、性质有关的对象；与数据之间联系有关的对象

数据结构是对系统静态特性的描述

(2)数据操作

定义：对数据库中各种对象(型)的实例(值)允许执行的操作以及有关的操作规则

数据操作的类型：查询；更新(插入、删除、修改)

数据模型对操作的定义：操作的确切含义；操作符号；操作规则(如优先级)；实现操作的语言

数据操作是对系统动态特性的描述

(3)数据的完整性约束条件

完整性规则：给定的数据模型中数据及其联系所具有的制约和储存规则

用以限定符合数据模型的数据库状态以及状态的变化，以保证数据的正确、有效、相容

数据模型对完整性约束条件的定义：反映和规定本数据模型必须遵守的基本的通用的完整性约束条件。例如在关系模型中，任何关系必须满足实体完整性和参照完整性两个条件，提供定义完整性约束条件的机制，以反映具体应用所涉及的数据必须 遵守的特定的语义约束条件

3.概念模型

用途：用于信息世界的建模；是现实世界岛机器世界的一个中间层次；是数据库设计的有力工具；数据库设计人员和用户之间进行交流的语言

对概念模型的基本要求：较强的语义表达能力；能够方便、直接地表达应用中的各种语义只是、简单、清晰、易于用户理解

(1)信息世界中的基本概念

实体(entity)：客观存在并可相互区别的事物称为实体。可以是具体的人事物或者抽象的概念

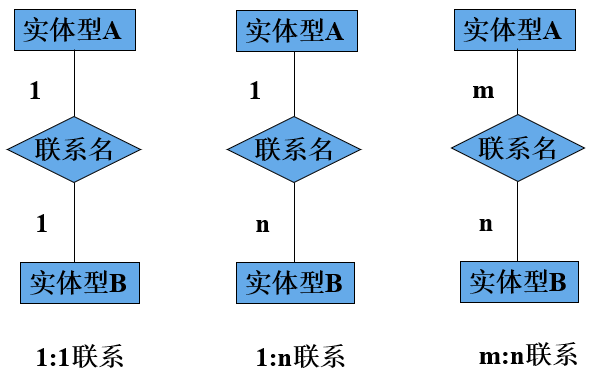
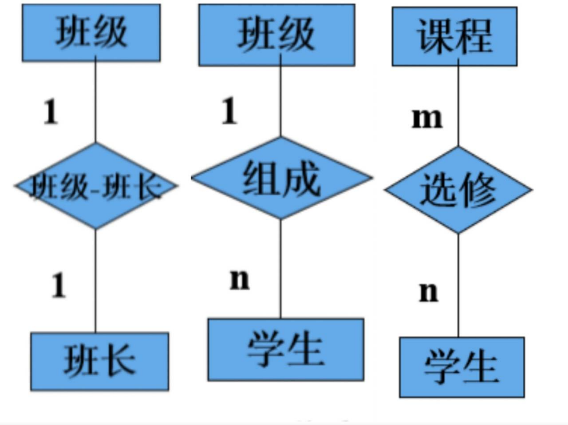
属性(attribute)：实体所具有的某一特性称为属性。一个实体可以由若干个特性来刻画

码(key)：唯一表示实体的属性集称为码

域(domin)：属性的取值范围称为该属性的域

实体型(entity type)：用实体名及其属性名集合来抽象和刻画同类实体称为实体型

实体集(entity set)：同一类型实体的集合称为实体集



联系(relationship)：现实世界中事物内部以及事务之间的联系在信息世界中反映为实体内部的联系和实体之间的联系；实体内部的连体通常是指组成实体的各属性之间的联系；实体之间的联系通常是指不同实体集之间的联系

(2)两个实体型之间的联系

用图形表示两个实体型之间的三类关系：

1)一对一联系(1:1)

实例：一个班级只有一个正班长，一个正班长只在一个班级任职

定义： 如果对于实体集A中的每一个实体，实体集B中至多有一个(也可以没有)实体与之联系，反之亦然，则称实体集A与实体集B具有一对一联系，记为1:1

2)一对多联系(1:n)

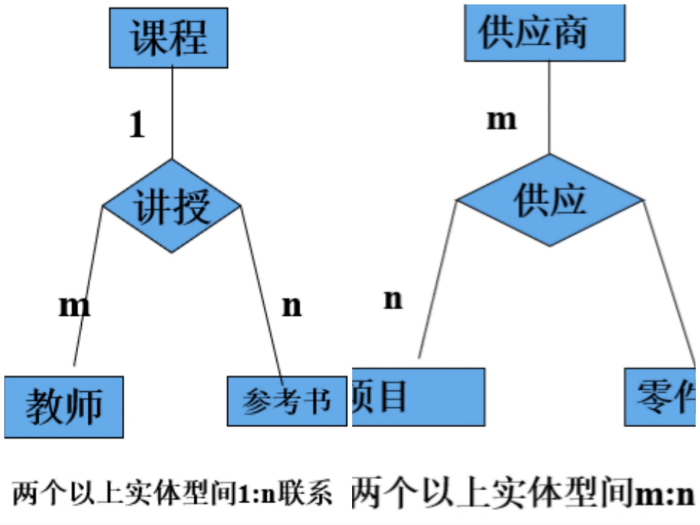
实例：一个班级中有若干名学生，每个学生只在一个班级中学习

定义：如果对于实体集A中的每一个实体，实体集B中有n个实体(n≥0)与之联系，反之，对于实体集B中的每一个实体，实体集A中至多只有一个实体与之联系，则称实体集A与实体集B有一对多联系，记为1:n

3)多对多联系(m:n)

实例：课程与学生之间的联系，一门课程同时又若干学生选修，一个学生可同时选修多门课程

定义：如果对于实体集A中的每一个实体，实体集B中有n个实体(n≥0)与之联系，反之，对于实体集B中的每一个实体，实体集A中也有m个实体(m≥0)与之联系，则称实体集A与实体B具有多对多联系，记为m:n

(3)两个以上实体型之间的联系

1)两个以上实体型一对多联系

若实体集E1，E2，...，En存在联系，对于实体集Ej(j=1，2，...，i-1，i+1，...，n)中的给 定实体，最多只和Ei中的一个实体相联系，则我们说Ei与E1，E2，...，Ei-1，Ei+1，...，En之间的联系是一对多的

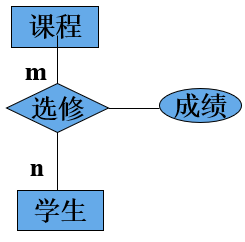
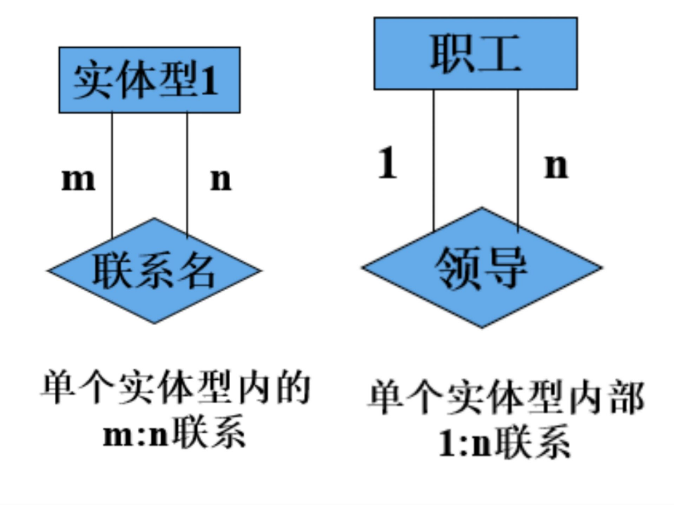
实例：课程、教师与参考书三个实体型；一门课程可以有若干个教师讲授，使用若干本参考书，每一个教师只讲授一门课程，每一本参考书只供一门课程使用

2)多个实体型之间的对一联系(略)

3)两个以上实体型之间的多对多联系

实例：供应商、项目、零件三个实体型，一个供应商可以供给多个项目多种零件，每个项目可以使用多个供应商供应的零件，每种零件可由不同供应商供给

(4)单个实体型内的联系



1)一对多联系

实例：职工实体型内部具有领导与被领导的联系，某一职工(干部)“领导”若干名职工，一个职工仅被另外一个职工直接领导，这是一对多的联系

2)一对一联系(略)

多对多联系

(5)概念模型的一种表示方法

实体--联系方法(E-R方法)：用E-R图来描述现实世界的概念模型；E-R方法也称为E-R模型

实体型：用矩形表示，矩形框内写明实体名

属性：用椭圆形表示，并用无向边将其与对应的实体连接起来

联系：

联系本身：用菱形表示，菱形框内写明联系名，并用无向边分别于有关实体连接起来，同时在无向边旁边表上联系的类型(1:1/1:n/m:n)

联系的属性：联系本身也是一种实体型，也可以有属性。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

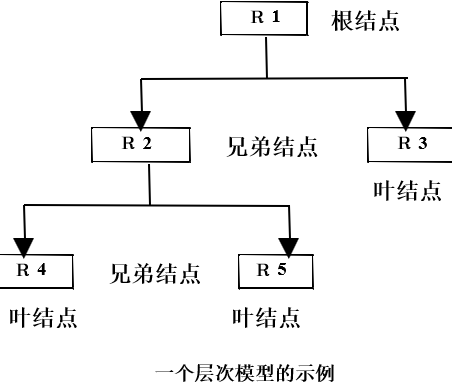
4.最常用的数据模型

非关系模型(层次模型；网状模型)；关系模型；面向对象模型；对象关系模型

5.层次模型

层次模型是数据库系统中最早出现的数据模型；层次数据库系统的典型代表是IBM公司1968年推出的IMS数据库管理系统；层次模型用树形结构来表示各类实体以及实体间的联系

(1)层次数据模型的数据结构



层次模型，满足下面两个条件的基本层次联系的集合为层次模型：有且只有一个节点没有双亲节点，这个节点称为根节点；根以外的其他节点有且只有一个双亲节点

层次模型中的几个术语：根节点，双亲节点，兄弟节点，叶节点

层次模型的特点：节点的双亲是唯一的；只能直接处理一对多的实体联系；每个记录类型可以定义一个排序字段，也称为码段；在层次模型中具有一定的存取路径，它只允许自顶向下的单项查询；没有一个子女记录值能脱离双亲记录值而独立存在

(2)多对多联系在层次模型中的表示

多对多联系在层次模型中的表示：用层次模型间接表示多对多联系

方法：将多对多联系分解成一对多联系(分解方法为冗余结点法)

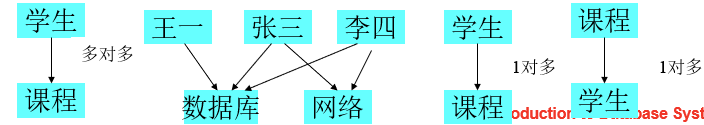
(3)层次模型的数据操纵与完整性约束

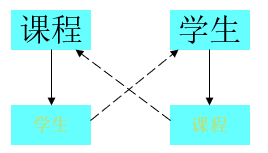
虚拟结点法(见右图)

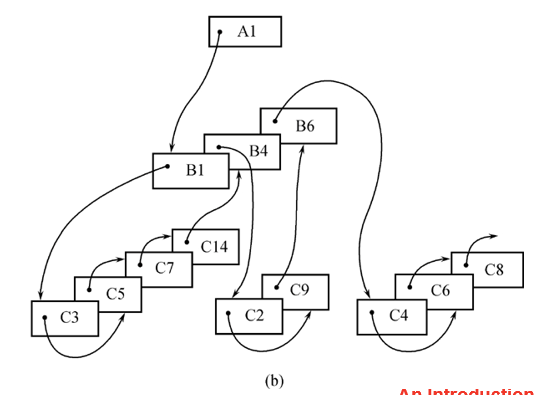
层次模型的数据操纵：查询、插入、删除、更新

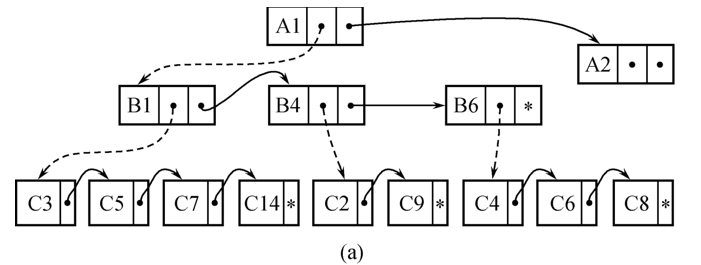
层次模型的完整性约束条件：无相应的双亲结点值就不能插入子女结点值；如果删除双亲结点值，则相应的子女节点也被同时删除；更新操作时，应更新所有的相应记录，以保证数据的一致性

(4)层次数据模型的存储结构









1)邻接法

按照层次树前序遍历的顺序把所有记录值依次邻接存放，即通过物理空间的位置相邻来实现层次顺序(放入每个结点后放入它的子女结点再放入它的兄弟结点)

2)链接法

用指针来反映数据之间的层次联系；子女-兄弟链接法；层次序列链接法

子女-兄弟链接法(a)：每次记录设两类指针，分别指向最左边的子女(每个记录型对应一个)和最近的兄弟

层次序列链接法(b)：按树的前序穿越顺序链接各记录值

(5)层次模型的优缺点

优点：层次模型的数据结构比较节点清晰；查询效率高，性能优于关系模型，不低于网状模型；层次数据模型提供了良好的完整性支持

缺点：多对多链接不自然；对插入和删除操作的限制多，应用程序的编写较为复杂；查询子女几点必须通过双亲结点；由于结构严密，层次命令趋于程序化

6.网状模型

网状数据库系统采用网状模型作为数据的组织方式

典型代表是DBTG系统：又称CODASYL(数据系统语言研究会)系统；70年代由DBTG(Data Base Task Group)提出的一个系统方案；奠定了数据库系统的基本概念、方法、技术

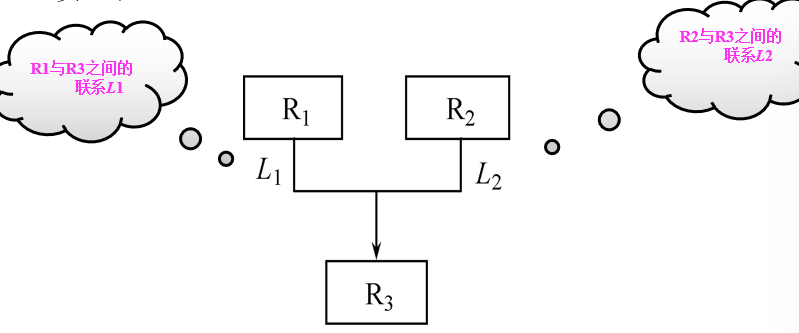
实际系统：Cullinet Software Inc.公司的 IDMS；Univac公司的 DMS1100；Honeywell公司的IDS/2；HP公司的IMAGE

(1)网状数据模型的数据结构

满足两个条件的基本层次联系的集合为网状模型：允许一个以上结点无双亲；一个以上结点可以有多于一个的双亲

表示方法(与层次数据模型相同):

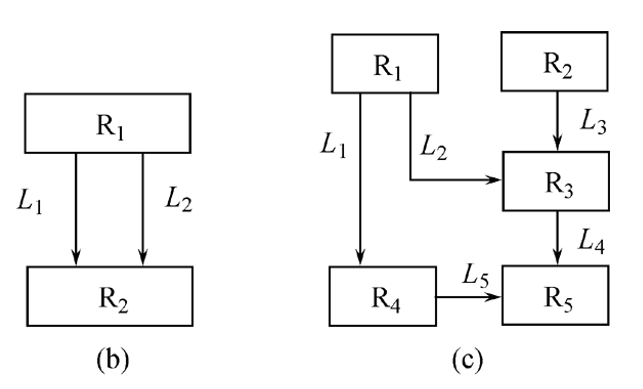
实体型：用记录类型描述；每个结点表示一个记录类型(实体)



属性：用字段描述；每个记录类型可包含若干个字段

联系：用结点之间的连线表示记录类型(实体)之间的一对多的父子联系

网状模型与层次模型的区别：网状模型允许多个节点没有双亲结点；网状模型允许结点有多个双亲结点；网状模型允许两个结点之间有多种联系；网状模型可以更直接地区描述现实世界；层次模型实际上是网状模型的一个特例



网状模型中子女结点与双亲结点的联系可以不唯一：要为每个联系命名，并指出与该联系有关的双亲记录和子女记录

多对多联系在网状模型中的表示：用网状模型间接表示多对多联系

方法：将多对多联系直接分解成一对多联系

(2)网状数据模型的操纵与完整性约束

网状数据库系统(如DBTG)对数据操纵加了一些限制，提供了一定的完整性约束

码：唯一标识记录的数据项的集合

一个联系中双亲记录与子女记录之间是一对多联系

支持双亲记录和子女记录之间的某些约束条件

(3)网状数据模型的存储结构

关键：实现记录之间的联系

常用方法：单向链接；双向链接；环状链接；向首链接

(4)网状数据模型的优缺点

优点：能够更为直接的描述现实世界，如一个节点可以有多个双亲结点；具有良好的性能，存取效率较高

缺点：结构比较复杂，而且随着应用环境的扩大，数据库的结构就变得越来越复杂，不利于最终用户掌握；DDL、DML语言复杂，用户不容易使用

7.关系模型

关系数据库系统采用关系模型作为数据的组织方式，建立在严格的数学概念的基础上

1970年美国IBM公司San Jose研究室的研究员E.F.Codd首次提出了数据库系统的关系模型

计算机厂商新推出的数据库管理系统几乎都支持关系模型

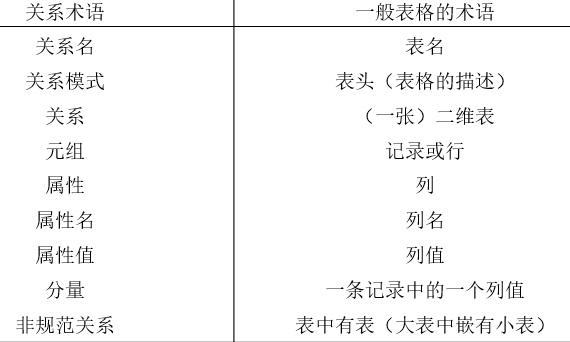
(1)关系数据模型的数据结构

在用户观点下，关系模型中的逻辑结构是一张规范化的二维表，它由行和列组成

关系：一个关系对应通常说的一张表

元组：表中的一行即为一个元组

属性：表中的一列即为一个属性，给每一个属性起一个名称即属性名

主码：表中的某个属性组，它可以唯一确定一个元组

域：属性的取值范围

分量：元组中的一个属性值

关系模式：对关系的描述；关系名(属性1，属性2，......，属性n)(如：学生(学号，姓名，年龄，性别，系，年级)

关系必须是规范化的，满足一定的规范条件

最基本的规范条件：关系的每一个分量必须是一个不可分的数据项，不允许表中还有表

(2)关系数据模型的操纵与完整性约束

关系模型中的数据操作都是集合操作，操作对象和操作结果都是集合(关系)，即若干元组的集合；操作：查询、插入、删除、更新

存取路径对用户隐蔽，用户只要指出“干什么”，不必仔细说明“怎么干”

关系的完整性约束条件：实体完整性；参照完整性；用户定义的完整性

(3)关系数据模型的存储结构

实体及实体间的联系都用表来表示；表以文件形式存储

有的DBMS一个表对应一个操作系统文件；有的DBMS自己设计文件结构

(4)关系数据模型的优缺点

优点：建立在严格的数学概念的基础上；概念单一(实体和各类联系都用关系来表示；对数据的检索结果也是关系)；关系模型的存取路径对用户透明(具有更高的数据独立性，更好的安全保密性；简化了程序员的工作和数据库开发建立的工作)

缺点：存取路径对用户透明导致查询效率往往不如非关系型数据模型；为提高性能必须对用户的查询请求进行优化，增加了开大DBMS的难度

(三)数据库系统结构

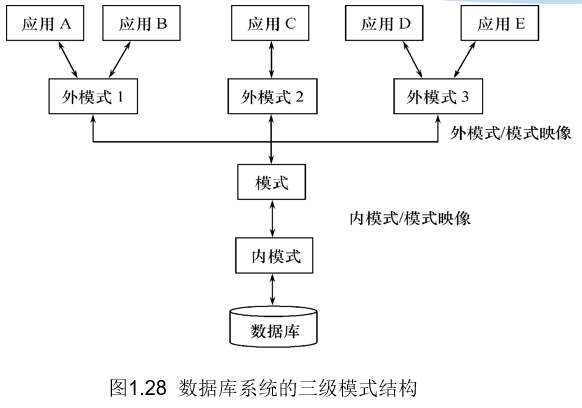
从数据库管理系统看，数据库系统通常采用三级模式结构，是数据库系统内部的系统结构

从数据库最终用户角度看(数据库系统外部的体系结构)，数据库系统的结构分为：单用户结构；主从式结构；分布式结构；客户/服务器；浏览器/应用服务器/数据库服务器多层结构等

1.数据库系统模式的概念

“型”和“值”的概念：型-->对某一数据的结构和属性的说明；值-->是型的一个具体赋值

例：学生记录型：(学号，姓名，性别，系别，年龄，籍贯)；一个记录值(900201，李明，男，计算机，22，江苏)

模式：数据库逻辑结构和特征的描述；是型的描述；反应的是数据的结构及其联系；模式是相对稳定的

实例：模式的一个具体值；反映数据库某一时刻的状态；同一模式可以有很多实例；实例对数据库中的数据的更新而变动

2.数据库系统的三级模式结构

模式；外模式；内模式

(1)模式(基本表)

模式(也称逻辑模式)：数据库中全体数据的逻辑结构和特征的描述；所有用户的公共数据视图，综合了所有用户的需求

一个数据库只有一个模式

模式的地位：是数据库系统模式结构的中间层；与数据的物理存储细节和硬件环境无关；与具体的应用程序、开发工具、高级程序设计语言无关

模式的定义：数据的逻辑结构(数据项的名字、类型、取值范围等)；数据之间的联系；数据有关的安全性完整性要求

(2)外模式(视图)

外模式(也称子模式或用户模式)：数据库用户(包括应用程序和最终用户)使用的局部数据的逻辑结构和特征的描述；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

外模式的地位：介于模式与应用之间；模式与外模式的关系-->一对多；一个数据库可以由多个外模式，反映了不同用户的应用需求、看待数据的方式、对数据保密的要求；对模式中同一数据，在外模式中的结构、类型、长度、保密级别等都可以不同

外模式与应用的关系：一对多；同一外模式也可以为某一用户的多个应用系统所使用；但一个应用程序只能使用一个外模式

外模式的用途：保证数据库安全性的一个有力措施；每个用户只能看见和访问所对应的外模式中的数据

(3)内模式(索引)

内模式(也称存储模式)：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示方法；记录的存储方式(顺序存储，B树结构存储，hash方法存储，堆存储等)；缩影的组织方式拉数据是否压缩；数据是否加密；数据存储记录结构的规定

一个数据库只能有一个内模式

3.数据库的二级映像功能与数据独立性

三级模式是对数据的三个抽象级别

二级映像在DBMS内部实现这三个抽象层次的联系和转换(映像都包含在每个名字组合的前一个模式中)

(1)外模式/模式映像

模式描述的是数据的全局逻辑结构；外模式描述的是数据的局部逻辑结构；同一个模式可以由任意多个外模式；每一个外模式，数据库系统都有一个外模式/模式映像，定义外模式与模式之间的对应关系；映像定义通常包含在各自外模式的描述中

保证数据的逻辑独立性：当模式改变时，数据库管理员修改有关的外模式/模式映像，使外模式保持不变；应用程序由数据的外模式编写的，从而应用程序不必改变，保证了数据与程序的逻辑独立性，简称数据的逻辑独立性

(2)模式/内模式映像

模式/内模式映像定义了数据全局逻辑结构域存储结构之间的对应关系；数据库中模式/内模式映像是唯一的；该映像通常包含在模式描述中

保证数据的物理独立性：当数据库的存储结构改变了(例如选用了另一种存储结构)，数据库管理员修改模式/内模式映像，使模式保持不变；应用程序不受影响，保证了数据与程序的物理独立性，简称数据的物理独立性

数据库模式，即全局逻辑结构，数据库的中心和关键；设计数据库模式结构时应首先确定数据库的逻辑模式

数据库的内模式：依赖于他的全局逻辑结构；将全局逻辑结构所定义的数据结构及其联系按照一定的物理存储策略进行组织，以达到较好的时间与空间效率

数据库的外模式：面向具体的应用程序；定义在逻辑模式之上；独立于存储模式与存储设备；当应用需求发生较大变化，相应外模式不能满足其视图要求时，该外模式就得做相应改动；设计外模式时应充分考虑到应用的扩充性

特定的应用程序：在外模式描述的数据结构上编制的；依赖于特定的外模式；与数据库的模式和存储结构独立；不同的应用程序又是可以共用一个外模式

数据库的二级映像：保证了数据库外模式的稳定性；从底层保证了应用程序的稳定性，除非应用需求本身发生变化，否则应用程序一般不需要修改

数据与程序之间的独立性，使得数据的定义和描述可以从应用程序中分离出去

数据的存取由DBMS管理：用户不必考虑存取路径等细节；简化了应用程序的编制；大大减少了应用程序的维护和修改

4.数据库系统的组成

数据库；数据库管理系统；应用系统；数据库管理员；硬件平台及数据库；软件；人员

(1)硬件平台及数据库

数据库系统对硬件资源的要求：

足够大的内存：操作系统；DBMS的核心模块；数据缓冲区；应用程序

足够大的外存：磁盘或磁盘阵列-->数据库；光盘、磁带-->数据备份

较高的通道能力，提高数据传送率

(2)软件

DBMS；支持DBMS运行的操作系统；与数据库接口的高级语言及其编译系统；以DBMS为核心的应用开发工具；为特定应用环境开发的数据库应用系统

(3)人员

1)数据库管理员：

决定数据库中的信息内容和结构；决定数据库的存储结构和存取策略；定义数据的安全性要求和完整性约束条件；监控数据库的使用和运行(周期性转储数据库包括数据文件和日志文件；系统故障恢复；介质故障恢复；监视审计文件)；数据库的改进和重组(性能监控和调优；定期对数据库惊进行重组织，以提高系统的性能；需求增加和改变时，数据库需要重构造)

2)系统分析员和数据库设计人员：

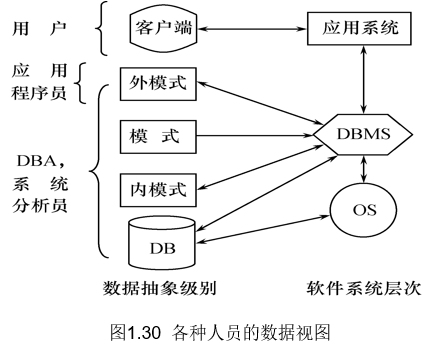
系统分析员负责应用系统的需求分析和规范说明；与用户及DBA协商，确定系统的硬软件配置；参与数据库系统的概要设计

数据库设计人员参加用户需求调查和统计分析，确定数据库中的数据，设计数据库各级模式

3)应用程序员

设计和编写应用系统的程序模块；进行调试和安装

4)用户

用户指最终用户，其通过应用系统的用户接口使用数据库

偶然用户：不经常访问数据库，但每次访问数据库时往往需要不同的数据库信息 -->企业或组织机构的高中级管理人员

简单用户：主要工作是查询和更新数据库-->银行的职员、机票预定人员、旅馆总台服务员

复杂用户：直接使用数据库语言访问数据库，甚至能够基于数据库管理系统的API编制自己的应用程序-->工程师、科学家、经济学家、科技工作者等

不同的人员设计不同的数据抽象级别，具有不同的数据视图

二、关系数据库

(一)关系数据结构及形式化定义

1.关系模型概述

(1)关系

1)概念

关系是单一的数据结构，现实世界的实体以及实体间的各种联系均用关系来表示

逻辑结构-->二维表：从用户角度，关系模型中数据的逻辑结构是一张二维表

建立在几何代数的基础上

域：一组具有相同数据类型的值的集合(如：整数；实数；介于某个取值范围的数；{'男','女'})

笛卡尔积：给定一组域D1,D2,......Dn，这些域中可以有相同的

D1,D2,......Dn的笛卡尔积为：D1×D2×…×Dn ＝ ｛(d1，d2，…，dn)｜di属于Di，i＝1，2，…，n｝

所有域的所有取值的一个组合；不能重复

元组：笛卡尔积中每一个元素(d1,d2,......dn)叫作一个n元组或者简称元组。如(张清玫，计算机专业，李勇)、(张清玫，计算机专业，刘晨)等都是元组

分量：笛卡尔积元素(d1,d2,......dn)中的每一个值di叫作一个分量。如张清玫、计算机专业、李勇、刘晨等都是分量

0

基数：若Di(i=1,2,...n)为有限集，其基数为mi(i=1,2,...,n)则D1×D2×…×Dn的基数M：见右图

笛卡尔积的表示方法：笛卡尔积可以表示为一个二维表；表中的每一行对应一个元组，表中的每一列对应一个域

2)关系

关系：D1×D2×…×Dn的子集叫作在域D1,D2,...,Dn上的关系，表示为R(D1,D2,...Dn)；R：关系名，n：关系的目或度

元组：关系中的每个元组是关系中的元组，通常用t表示

单元关系和二元关系：当n=1时，该关系为单元关系/一元关系；当n=2时，该关系为二元关系

关系的表示：关系也是一个二维表，表的每一行对应一个元组，表的每一列对应一个域

属性：关系中不同列可以对应相同的域，为了加以区分，必须对每一列起一个名字，称为属性，n目关系必有n个属性

码:

候选码：若关系中的某一属性组的值能唯一标识一个元组，而其子集不能，则称该属性组为候选码(简单的情况下，候选码只包含一个属性)

全码：最极端的情况-->关系模式中的所有属性组是这个管子模式的候选码，称为全码

主码：若一个关系有多个候选码，则选定其中一个为主码

主属性：候选码的诸属性称为主属性，不包含在任何候选码中的属性称为非主属性或非码属性

D1，D2，…，Dn的笛卡尔积的某个子集才有实际含义

3)三类关系

基本关系(基本表或基表)：实际存在的表，是实际存储数据的逻辑表示

查询表：查询结果对应的表

视图表：由基本表或其他视图表导出的表

查询表和视图表是虚表，不对应实际存储的数据

4)基本关系的性质

列是同质的，不同的列可以出自同一个域(其中的每一列称为一个属性，不同的属性要给予不同的属性名)；

列的顺序无所谓，列的次序可以任意交换；

任意两个元组的候选码不能相同；

行的顺序无所谓，行的次序可以任意交换；

分量必须取原子值(规范条件中的最基本的一条)

(2)关系模式

关系模式是型；关系是值；关系模式是对关系的描述(元组集合的结构-->属性构成，属性来自的域，属性与域之间的映像关系；元组语义以及完整性约束条件；属性间的数据依赖关系集合)

关系模式可以形式化地表示为：R(U，D，DOM，F)

R：关系名；U：组成该关系的属性名集合；

D：属性组U中属性所来自的域；DOM：属性向域的映象集合

F：属性间的数据依赖关系集合

关系模式通常可以简记为：R (U)或R (A1，A2，…，An)

R：关系名；A1，A2，…，An： 属性名

注：域名及属性向域的映象常常直接说明为属性的类型、长度

关系模式是对关系的描述，是静态的稳定的；关系是关系模式在某一时刻的状态或者内容，是动态的随时间不断变化的；关系模式和关系往往统称为关系。二者通过长下文加以区别

(3)关系数据库

在一个给定的应用领域中，所有关系的集合构成一个关系数据库

关系数据库的型与值

型：关系数据库模式对关系数据库的描述；关系数据库模式包括：若干域的定义，在这些域上定义的若干关系模式

值：关系模式在某一时刻对应的关系的集合，简称为关系数据库

2.关系操作

(1)基本关系操作

1)常用的关系操作

查询：选择、投影、连接、除、交、并、差

数据更新：插入、删除、修改

查询的表达能力是其中最主要的部分

选择、投影、并、差、笛卡尔积是5种基本操作

2)关系操作的特点

集合操作方式：操作的对象和结果都是集合，一次一集合的方式

(2)关系数据库语言的分类

关系代数语言：用对关系的运算来表达查询要求。代表：ISBL

关系演算语言：用谓词来表达查询要求

元组关系演算语言：谓词变元的基本对象是元组变量。代表：APLHA，QUEL

域关系演算语言：谓词变元的基本对象是域变量。代表：QBE

具有关系代数和关系演算双重特点的语言：代表：SQL

3.关系的完整性

(1)三类完整性约束

实体完整性和参照完整性：关系模型必须满足的完整性约束条件，称为关系的两个不变形，应该有关系系统自动支持

用足定义的完整性：应用邻域需要遵循的约束条件，体现了具体领域中的语义约束

(2)实体完整性

实体完整性规则：若属性A是基本关系R的主属性，则属性A不能取空值

说明：

实体完整性规则是针对基本关系而言的，一个基本表通常对应现实世界的一个实体集

现实世界中的实体是可区分的，即他们只有某种唯一性标识

关系模型中以主码作为唯一标识

主码中的属性即主属性不能取空值：主属性取空值，就说明存在某个不可标识的实体，即存在不可区分的实体，这与第二点矛盾，因此这个规则称为实体完整性

(3)参照完整性

1)关系间的引用

在关系模型中实体及实体间的联系都是用关系来描述的，因此可能存在着关系与关系间的引用

例：S(SNO,SNAME,SSEX,CNO)；SC(CNO,SCNAME) 在学生实体中学号是主码，在专业实体中，课程号是主码，学生关系中引用了专业关系的主码“专业号(CNO)”学生关系中的“专业号”必须是确实群在的专业的专业号，即专业关系中必须有该专业的记录

在本例中，学生关系是参照关系，专业关系是被参照关系，专业号是学生关系的外码

2)外码

设F是基本关系R的一个或一组属性，但不是关系R的码，如果F与基本关系S的主码Ks相对应则称F是基本关系R的外码，基本关系R为参照关系，基本关系S为被参照关系或目标关系

关系R和S不一定是不同的关系；目标关系S的主码Ks和参照关系的外码F必须定义在同一个(或一组)域上；外码并不一定要与相应的主码同名；当外码与相应的主码属于不同关系时，往往去相同的名字，以便于识别

参照完整性规则：若属性(或属性组)F是基本关系R的外码，它与基本关系S的主码Ks相对应(关系R和S不一定是不同的关系)则对于R中的每个元组在F上的值必须为：空值(F的每个属性值均为空值)或者等于S中某个元组的主码值

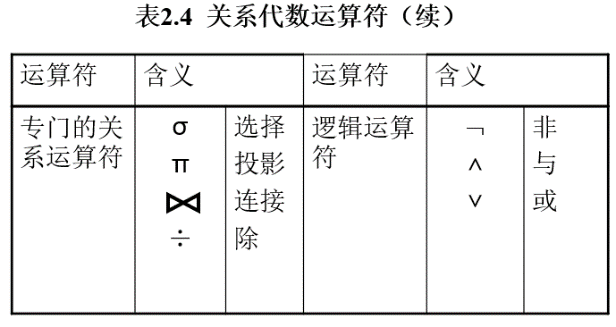
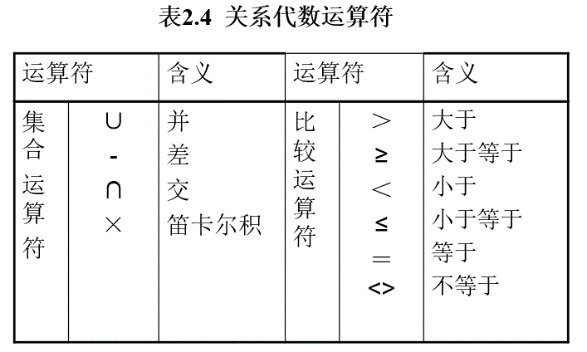
(4)用户定义的完整性

针对某一具体关系数据库的约束条件，反映某一具体应用所涉及的数据必须满足的语义要求

关系模型应提供定义和检验这类完整性的机制，以便于用同一的系统的方法处理他们，而不要由应用程序承担这一功能

4.关系代数

(1)概述



(2)传统的集合运算

1)并

R和S具有相同的目n(即两个关系都有n个属性)

R U S仍为n目关系，由数据R或数据S的元组组成  R∪S = { t|t ∈ R∨t ∈ S }

2)差

R和S具有相同的目n，相应的属性取自同一个域

R - S仍为n目关系，由属于R二不属于S的所有元组组成 R-S ={ t|t∈R∧t\∈S }

3)交

R和S具有相同的目n，相应的属性取自同一个域

R∩S仍为n目关系，由既属于R又属于S的元组组成      R∩S={t|t∈R∧t∈S}

R∩S = R –(R-S)

4)笛卡尔积

严格地讲是广义的笛卡尔积

R：n目关系，k1个元组 S：m目关系，k2个元组

R×S ：列：(n+m)列元组的集合，元组的前n列是关系R的一个元组，后m列是关系S的一个元组，行：k1×k2个元组

R×S = {tr ts |tr ∈R ∧ ts\∈S }

(3)专门的关系运算

1)引入符号

R，t∈R，t[Ai]：设关系模式为R(A1,A2,...,An)，它的一个关系设为R，t∈R表示t是R的一个元组，t[Ai]表示元组t中相应于属性Ai的一个分量

A，t[A]， A-(横线在上面)：若A={Ai1,Ai2,...,Aik}，其中Ai1,Ai2,...,Aik是A1,A2,...An中的一部分，则A称为属性列或属性组；t[A]=(t[Ai1]，t[Ai2]，…，t[Aik])表示元组t在属性列A上诸分量的集合；A-则表示{A1,A2,...,An}中去掉{Ai1,Ai2,...,Aik}后剩余的属性组

tr-ts：R为n目关系，S为m目关系    tr ÎR，tsÎS， tr-ts称为元组的连接。tr-ts是一个n + m列的元组，前n个分量为R中的一个n元组，后m个分量为S中的一个m元组。

象集Zx：给定一个关系R(X,Z)，X和Z为属性组。当t[X]=x时，x在R中的象集为Zx={t[Z]|t∈R，t[X]=x}，他表示R中属性组X上值为X的诸元组在Z上分量的集合

2)选择(对行进行选取)

选择又称为限制：σF(R) F(R)：可以是具体的属性σsage<20，也可以是列号σ4<20

选择的含义：在关系R中选择满足给定条件的诸元组       σF(R) = {t|tÎR∧F(t)= '真'}

F：选择条件，是一个逻辑表达式，基本形式为：X1θY1

选择运算是从关系R中选取使逻辑表达式F为真的元组，是从行的角度进行的运算

3)投影(对列进行选取)

投影：πA(R) A(R)：可以是具体的属性πsname,sdept，也可以是列号π2,5

从R中选择出若干属性列组成新的关系，πA(R) = { t[A] | tÎR }，  A：R中的属性列

投影操作主要是从列的角度进行运算，投影之后不仅取消了原关系中的某些列，(选取了某些列)，还可能取消某些元组(避免重复行)

4)连接(也称为θ连接)

连接：从两个关系的笛卡尔积中选取属性间满足一定条件的元组

0

A和B：R和S上目数相等且可比的属性组，θ：比较运算符

连接运算：从R和S的广义笛卡尔积R×S中选取(R关系)在A属性组上的值与(S关系)在B属性组上 的值满足比较关系θ的元组

等值连接

θ为=的连接运算，从关系R和S的广义笛卡尔积中选取A、B属性组值相等的元组

0

自然连接

一种特殊的等值连接，两个关系中进行比较的冯丽娜个必须是相同的属性组，在结果中把重复的属性列去掉，R和S具有相同的属性组

0

一般的连接操作是从行的角度进行运算的，自然连接还需要取消重复的列，所以是同时从行和列的角度进行运算

两个关系R和S在做自然连接时，选择两个关系在公共属性上值相等的元组构成新的关系，此时，关系R中某些元组有可能在S中不存在公共属性上值相等的元组，从而造成R中的这些元组在操作时被舍弃了，同样，S中某些元组也可能被舍弃，这些被舍弃的元组称为悬浮元组

外连接：如果把舍弃的元组也保存在结果关系中，而在其他属性上填空值，这种连接就叫做外连接

左外连接：如果只把左边关系R中要舍弃的元组保留就叫做左外连接

右外连接：如果只把右边关系S中要舍弃的元组保留就叫做左外连接

5)除

给定关系R(X,Y)和S(Y,Z)其中X,Y,Z为属性组。R中的Y与S中的Y可以有不同的属性名，但必须出资相同的域集，R与S的除运算得到一个新的关系P(X)，P是R中满足下列条件的元组在X属性列上的投影：元组在X上分量值的象集Yx包含S在Y上投影的集合记作：(Yx：X在R中的象集，x=tr[X])

0

除操作是同时从行和列角度进行运算

e.g.：在关系R中，A可以取四个值{a1，a2，a3，a4}，a1的象集为 {(b1，c2)，(b2，c3)，(b2，c1)}，a2的象集为 {(b3，c7)，(b2，c3)}，a3的象集为 {(b4，c6)}，a4的象集为 {(b6，c6)}，S在(B，C)上的投影为，{(b1，c2)，(b2，c1)，(b2，c3) }，只有a1的象集包含了S在(B，C)属性组上的投影，所以 R÷S ={a1}

5.关系演算

以数理逻辑中的谓词演算为基础，按谓词变元不同，进行分类：

元组关系演算：以元组变量作为谓词变元的基本对象；元组关系演算语言：ALPHA

域关系演算：以域变量作为谓词变元的基本对象；域关系演算语言：QBE

(1)元组关系演算语言ALPHA

检索：GET 更新：PUT,HOLD,UPDATE,DELETE,DROP

语句格式：GET 工作空间名(定额)(表达式1)：操作条件 DOWN/UP表达式2

定额：检索的元组个数。格式：数字

表达式1：指定语句的操作对象。格式：关系名/关系名.属性名/元组变量.属性名/集函数

操作条件：将操作结果限定在满足条件的元组中。格式：逻辑表达式

表达式2：指定排序方式。格式：关系名.属性名/元组变量.属性名/集函数

1)简单检索：GET 工作空间名 (表达式1)

e.g.查询所有被选修的课程号码：GET W (SC.Cno) 查询所有学生的数据：GET W (STUDENT)

2)限定检索：GET 工作空间名 (表达式1)：操作条件

e.g.查询信息系中年龄小于20的学生的学号年龄：

GET W (STUDENT.Sno,STUDENT.SAGE) : STUDENT.Sdept='IS'^STUDENT.Sage<20

3)带排序的检索：GET 工作空间名 (表达式1)：操作条件 DOWN/UP 表达式2

e.g.查询计算机科学系(CS)学生的学号、年龄，结果按年龄降序排序: GET W (Student.Sno，Student.Sage) : Student.Sdept='CS‘ DOWN Student.Sage

4)带定额的检索：GET 工作空间名(定额)(表达式1)：操作条件 DOWN/UP 表达式2

e.g.取出一个信息系学生的学号：GET W (1) (Student.Sno):Student.Sdept='IS'

查询信息系年龄最大的三个学生的学号及其年龄，结果按年龄降序排序：GET W (3) (Student.Sno，Student.Sage):Student.Sdept='IS' DOWN Student.Sage

5)用元组变量的检索

元组变量的含义：表示可以在某一关系范围内变化(也称为范围变量Range Variable)

元组变量的用途：简化关系名：设一个较短的名字的元组变量来代替较长的关系名；操作条件中使用量词时必须使用元组变量

定义元组变量：格式：RANGE 关系名 变量名；一个关系可以设多个元组变量

6)用存在量词的检索(用$表示存在量词)

操作条件中使用量词时必须用元组变量

e.g.查询选修2号课程的学生名字：RANGE SC  X GET W (Student.Sname): .Sno=Student.Sno∧X.Cno='2')

查询选修了这样课程的学生学号，其直接先行课是6号课程：RANGE Course  CX   GET W  (SC.Sno):   $CX (CX.Cno=SC.Cno∧CX.Pcno='6')

查询至少选修一门其先行课为6号课程的学生名字：

RANGE Course CX SC SCX GET W (Student.Sname): $SCX (SCX.Sno=Student.Sno∧$CX (CX.Cno=SCX.Cno∧CX.Pcno='6'))

前束范式形式： GET W (Student.Sname)：$SCX$CX (SCX.Sno=Student.Sno∧CX.Cno=SCX.Cno∧CX.Pcno='6')

7)带有多个关系的表达式的检索

查询成绩为90分以上的学生名字与课程名字： RANGE SC SCX GET W(Student.Sname，Course.Cname): $SCX (SCX.Grade≥90 ∧ SCX.Sno=Student.Sno∧Course.Cno=SCX.Cno)

8)用全称量词的检索(用#表示全称量词，~表示非)

查询不选1号课程的学生名字 RANGE SC SCX GET W (Student.Sname):#SCX (SCX.Sno≠Student.Sno∨SCX.Cno≠'1')

用存在量词表示：RANGE SC SCX GET W (Student.Sname):~$SCX (SCX.Sno=Student.Sno∧SCX.Cno='1')

9)用两种量词的检索

查询选修了全部课程的学生姓名： RANGE Course CX SC SCX GET W (Student.Sname): #CX$SCX (SCX.Sno=Student.Sno∧SCX.Cno=CX.Cno)

10)用蕴含的检索(#表示任意，→表示蕴含)

查询最少选修了200215122学生所选课程的学生学号：

RANGE Couse CX SC SCX SC SCY

GET W (Student.Sno):

#CX($SCX(SCX.Sno=‘200215122'∧SCX.Cno=CX.Cno)→SCY(SCY.Sno=Student.Sno∧SCY.Cno= CX.Cno))

11)聚集函数

常用聚集函数或者内部函数：

COUNT：对元组计数 TOTAL/MAX/MIN/AVG：求总和/最大值/最小值/平均值

e.g.查询学生所在系的数目：GET W  ( COUNT(Student.Sdept) ) COUNT函数在计数时会自动排除重复值

查询信息系学生的平均年龄：GET  W (AVG(Student.Sage):Student.Sdept='IS’ )

2)更新操作

a.修改

用HOLD语句将要修改的元组从数据库中读取到工作空间中(HOLD语句是带上并发控制的GET语句):

HOLD 工作空间名(表达式1)：操作条件

用宿主语言修改工作空间中元组的属性，再用UPDATE语句将修改后的元组送回数据库中

UPDATE 工作空间名

e.g.把200215121学生从计算机科学系转到信息系。

    HOLD W (Student.Sno,Student.Sdetp):Student.Sno=‘200215121'(从Student关系中读出95007学生的数据)

MOVE 'IS' TO W.Sdept      (用宿主语言进行修改)

UPDATE W                   (把修改后的元组送回Student关系)

但：对主码进行修改的操作是不允许的，要修改主码值，只能先删除该元组之后再插入新主码值的元组

将学号200215121改为200215126

HOLD W (Student): Student.Sno=‘200215121'

DELETE W

MOVE ‘200215126' TO W.Sno

MOVE '李勇' TO W.Sname

MOVE '男' TO W.Ssex

MOVE '20‘ TO W.Sage

MOVE 'CS' TO W.Sdept

PUT W (Student)

b.插入

用宿主语言在工作空间中建立新元组

用PUT语句把该元组存放入指定关系中：PUT 工作空间名(关系名)

PUT语句只对一个关系操作，关系演算中的聚集函数

e.g.学校新开设了一门2学分的课程“计算机组织与结构”，其课程号为8，直接先行课为6号课程。插入该课程元组

MOVE '8' TO W.Cno

MOVE '计算机组织与结构' TO W.Cname

MOVE '6' TO W.Cpno

MOVE '2' TO W.Ccredit

PUT W (Course)

c.删除

用HOLD语句把要删除的元组从数据库中读到工作空间中

用DELETE语句删除该元组：DELETE 工作空间名

e.g.200215125学生因故退学，删除该学生元组

HOLD W (Student): Student.Sno=‘200215125'

DELETE W

删除全部学生

HOLD W (Student)

DELETE W

为保证参照完整性 ，删除Student中元组时相应地要删除SC中的元组

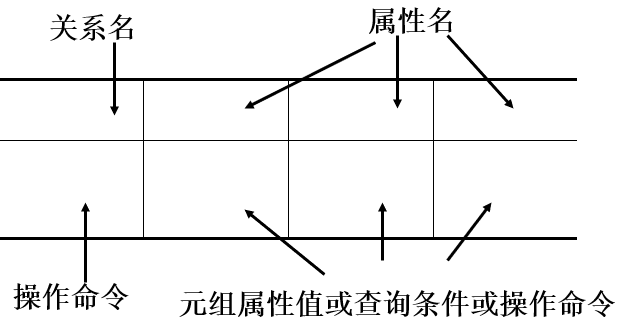
HOLD W (SC)

DELETE W

(2)域关系演算语言QBE

QBE(Query By Example)：基于屏幕表格的查询语言；查询要求：以填写表格的方式构造查询；

用示例元素(域变量)来表示查询结果可能的情况；查询结果：以表格形式显示



此段内容看ppt或者看书，与前后的关联不大，也较为简单

示例元素：即域变量，一定要加下划线。例元素是这个域中可能的一个值，它不必是查询结果中的元素

打印操作符P. 实际上是显示

查询条件：可使用比较运算符＞，≥，＜，≤，＝和≠，其中＝可以省略

升序排序：对查询结果按某个属性值的升序排序，只需在相应列中填入“AO.”

降序排序：按降序排序则填“DO.”

多列排序：如果按多列排序，用“AO(i).”或“DO(i).”表示，其中i为排序的优先级，i值越小，优先级越高

三、关系数据库标准语言SQL

(一)概述

SQL(Structured Query Language)：结构化查询语言，是关系数据库的标准语言，是一个通用的、功能极强的关系数据库语言

SQL特点：

综合统一：集数据定义语言(DDL)，数据操纵语言(DML)，数据控制语言(DCL)功能于一体；可以独立完成数据库生命周期的全部活动(定义关系模式，插入数据，建立数据库；对数据库中的数据进行查询和更新；数据库重构和维护；数据库安全性、完整性控制)；用户数据库投入运行后，可根据需要随时逐步修改模式，不影响数据的运行；数据操作符统一

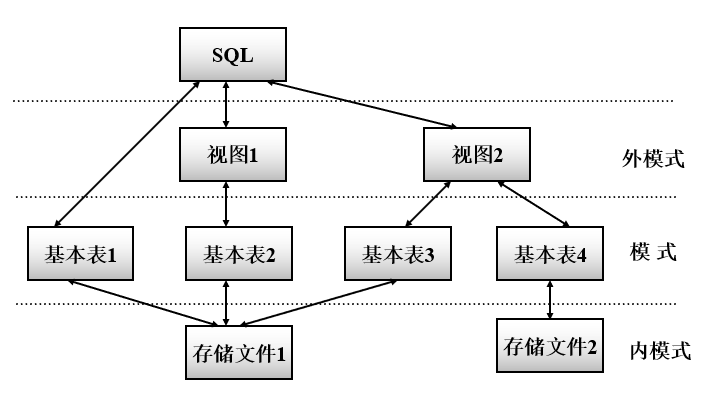
高度非过程化：非关系数据模型的数据操纵语言“面向过程”，必须指定存取路径；SQL只要提出“做什么”，无需了解存取路径；存取路劲的选择以及SQL的操作过程由系统自动完成

面向集合的操作方式：非关系数据模型采用面向记录的操作方式，操作对象是一条记录；SQL采用集合操作方式(操作对象、查找结果可以是元组的集合；一次插入、删除、更新操作的对象可以是元组的集合)

以同一种语法结构提供多种使用方式：SQL是对的语言(能够独立的用于联机交互的使用方式)；SQL是嵌入式语言(SQL能够嵌入到高级语言如C，C++，Java)程序中，供程序员设计程序时使用

语言简洁：SQL的核心功能只有9个动词：数据查询(SELECT)，数据定义(CREATE，DROP，ALTER)，数据操纵(INSERT，UPDATE，DELETE)，数据控制(GRANT，REVOKE)

SQL支持关系数据库三级模式结构



基本表：本身独立存在的表，SQL中一个关系就对应一个基本表，一个(或多个)基本表对应一个存储文件，

一个表可以带若干索引

存储文件：逻辑结构组成了关系数据库的内模式，物理结构是任意的，对用户透明

视图：从一个或几个基本表导出的表，数据库中只存放视图的定义而不存放视图对应的数据，视图是一个虚表，用户可以在视图上再定义视图

(二)学生-课程数据库

学生-课程模式 S-T : 学生表：Student(Sno,Sname,Ssex,Sage,Sdept)

课程表：Course(Cno,Cname,Cpno,Ccredit 学生选课表：SC(Sno,Cno,Grade)

(三)数据定义

SQL的数据定义功能：模式定义、表定义、视图和索引定义



1.模式的定义和删除

定义一个学生-课程模式S-T： CREATE SCHEMA “S-T” AUTHORIZATION WANG;

为用户WANG定义了一个模式S-T

CREATE SCHEMA AUTHORIZATION WANG； 隐含为模式名WANG， 如果没有指定，那么隐含为WANG

定义模式实际上定义了一个命名空间；在这个空间中可以定义该模式包含的数据库对象，如基本表、视图、索引等；在CREATE SCHEMA中可以接受CREATE TABLE，CREATE VIEW 和GRANT子句，CREATE SCHEMA AUTHORIZATION [||]

CREATE SCHEMA TEST AUTHORIZATION ZHANG

CREATE TABLE TAB1(COL1 SMALLINT，

COL2 INT，COL3 CHAR(20)， COL4 NUMERIC(10，3)， COL5 DECIMAL(5，2) )；

为用户ZHANG创建了一个模式TEST，并在其中定义了一个表TAB1。

删除模式

DROP SCHEMA

CASCADE(级联)：删除模式的同时把该模式中的所有数据库对象全部删除

RESTRICT(限制)：如果该模式中定义了下属的数据库对象(表视图等)则拒绝删除该删除语句的执行，当该模式中没有任何下属的对象是才能执行

DROP SCHEMA ZHANG CASCADE；

删除模式ZHANG，同时该模式中定义的表TAB1也被删除

2.基本表的定义删除修改

(1)定义基本表

CREATE TABLE ([ ]，[ ]，… [ ] )；

如果完整性约束条件涉及到该表的多个属性列，则必须定义在表级上，否则既可以定义在列级也可以定义在表级。

建立“学生”表Student，学号是主码，姓名取值唯一。

CREATE TABLE Student(

Sno CHAR(9) PRIMARY KEY， /\* 列级完整性约束条件\*/

Sname CHAR(20) UNIQUE， /\* Sname取唯一值\*/

Ssex CHAR(2)，

Sage SMALLINT，

Sdept CHAR(20) )；

建立一个“课程”表Course

CREATE TABLE Course(

Cno CHAR(4) PRIMARY KEY，

Cname CHAR(40)，

Cpno CHAR(4) ，

Ccredit SMALLINT，

FOREIGN KEY (Cpno) REFERENCES Course(Cno)); (CPNO是外码，被参照表是Course，被参照列是Cno)

建立一个“学生选课”表SC

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)

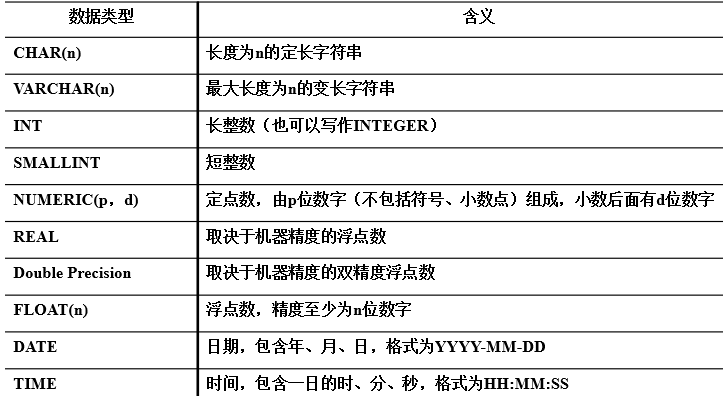
/\* 表级完整性约束条件， Cno是外码，被参照表是Course\*/);

(2)数据类型

SQL中的域的概念用数据类型实现

定义表的属性时，需要指明它的数据类型以及长度

选用合适的数据类型：取值范围 + 要做的运算



(3)模式与表

每一个基本表都属于某一个模式，一个模式包含多个基本表

定义基本表所属模式：

方法一：在表名中明显地给出模式名：

Create table "S-T".Student(......);/\*模式名为S-T\*

Create table "S-t".Course(......);

方法二：在创建模式语句中同时创建表

方法三：设置所属的模式

创建基本表(其他数据库对象也一样)时，若没有指定模式，系统会根据搜索路径来确定该对象所属的模式，RDBMS会使用模式列表中的第一个存在的模式作为数据库对象的模式名，若搜索路径中的模式名都不存在，系统将给出错误

显示当前的搜索路径：SHOW search\_path；

搜多路径的当前默认值：$user，PUBLIC

DBA用户可以设置搜索路径，然后定义基本表

SET search\_path TO "S-T"，PUBLIC

Create table Student(......)；

结果建立了S-T.Student基本表

RDMS发现搜索路径中第一个模式名S-T存在，就把该模式作为基本表Student所属的模式

(4)修改基本表

ALTER TABLE

[ ADD [ 完整性约束 ] ]

[ DROP ]

[ ALTER COLUMN ]；

e.g.向Student表增加“入学时间”列，其数据类型为日期型。

ALTER TABLE Student ADD S\_entrance DATE； 不论基本表中原来是否已有数据，新增加的列一律为空值。

将年龄的数据类型由字符型(假设原来的数据类型是字符型)改为整数。

ALTER TABLE Student ALTER COLUMN Sage INT；

增加课程名称必须取唯一值的约束条件。

ALTER TABLE Course ADD UNIQUE(Cname);

(5)删除基本表

DROP TABLE [RESTRICT|CASCADE]

RESTRICT：删除表是有限制的，要删除的表不能被其他表的约束所引用，如果存在依赖该表的对象，该表不能被删除

CASCADE：删除该表没有限制，在删除基本表的同时，相关的依赖对象一起删除

e.g.删除Student表

DROP TABLE Student CASCADE；

基本表定义被删除，数据别删除；表上建立的索引、视图、触发器等一般也会被删除

若表上建有视图，选择RESTRICT时表不能删除

CREATE VIEW IS\_Student

AS

SELECT Sno，Sname，Sage

FROM Student

WHERE Sdept='IS'；

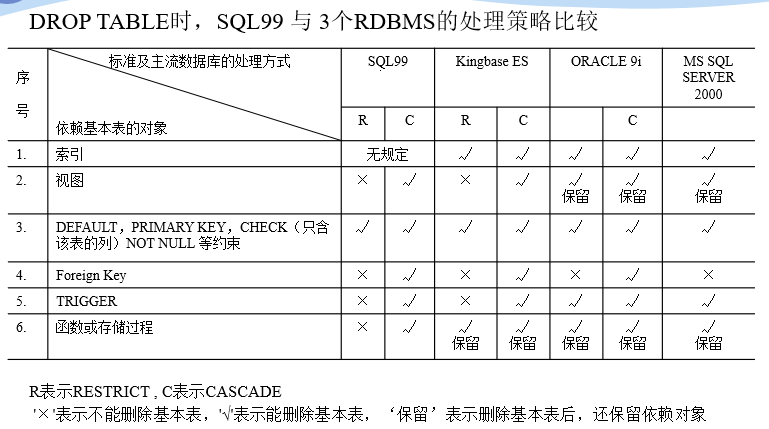
DROP TABLE Student RESTRICT;

--ERROR: cannot drop table Student because other objects depend on it

如果选择CASCADE时可以删除表，视图也自动被删除

DROP TABLE Student CASCADE; --NOTICE: drop cascades to view IS\_Student

SELECT \* FROM IS\_Student; --ERROR: relation " IS\_Student " does not exist



3.索引的建立与删除

建立索引的目的：加快查询速度

可以建立索引人：DBA或者表的属主(建立表的人)

DBMS一般会自动建立以下列上的索引：PRIMARY KEY UNIQUE

维护索引：DBMS自动完成

使用索引：DBMS自动选择是否使用索引以及使用哪些索引

RDBMS中索引一般采用B+树、HASH索引来实现：B+树索引有动态平衡的优点；HASH索引有查找速度快的优点

采用B+树还是HASH索引由具体的RDBMS来决定

索引是关系数据库的内部实现技术，属于内模式的范畴

CREATE INDEX语句定义索引时，可以定义索引是唯一索引、非唯一索引或聚簇索引

(1)建立索引

语句格式：CREATE [UNIQUE][CLUSTER] INDEX ON([][][]...)

UNIQUE-->唯一索引 CLUSTER-->聚簇索引

CREATE CLUSTER INDEX Stusname ON Student(Sname)；

在Student表的Sname(姓名)列上建立一个聚簇索引

在最经常查询的列上建立聚簇索引以提高查询效率，一个基本表上最多只能建立一个聚簇索引，经常更新的列不宜建立聚簇索引

e.g.为学生-课程数据库中的Student，Course，SC三个表建 立索引。

CREATE UNIQUE INDEX Stusno ON Student(Sno)；Student表按学号升序建唯一索引

CREATE UNIQUE INDEX Coucno ON Course(Cno)；Course表按课程号升序建唯一索引

CREATE UNIQUE INDEX SCno ON SC(Sno ASC，Cno DESC)；SC表按学号升序和课程号降序建唯一索引

(2)删除索引

语句格式：DROP INDEX 删除索引时，系统会从数据 字典中删去有关该索引的描述

e.g.删除Student表的Stusname索引 DROP INDEX Stusname；

(四)数据查询

语句格式：

SELECT [ALL|DISTINCT] [，] …

FROM [， ] …

[ WHERE ]

[ GROUP BY [ HAVING ] ]

[ HAVING ]

[ ORDER BY [ ASC|DESC ] ]；

1.单表查询

查询仅涉及一个表：选择表中的若干列；选择表中的若干元组；ORDER BY子句；聚集函数；GROUP BY子句

(1)选择表中的若干列

1)查询指定列

e.g.查询全体学生的学号与姓名： SELECT Sno，Sname FROM Student；

查询全体学生的姓名、学号、所在系： SELECT Sname，Sno，Sdept FROM Student；

2)查询全部列

选出所在属性列：在SELECT关键字后面列出所有列名/将指定为\*

e.g.查询全体学生的详细记录：SELECT  Sno，Sname，Ssex，Sage，Sdept FROMStudent 或 SELSCT \* FROM Student;

c.查询经过计算的值

SELECT 子句的可以为：算术表达式/字符串常量/函数/列别名

e.g.查全体同学的姓名及其出生年份：SELECT Sname，2023-Sage FROM Student；/\*当年年份减学生年龄\*/

查询全体学生的姓名、出生年份和所有系，要求用小写字母表示所有系名：SELECT Sname，‘Year of Birth: '，2004-Sage，ISLOWER(Sdept) FROM Student；

使用列别名改变查询结果的列标题: SELECT Sname NAME，'Year of Birth: ’ BIRTH，2000-Sage RTHDAY，LOWER(Sdept) DEPARTMENT FROM Student；

(2)选择表中的若干元组

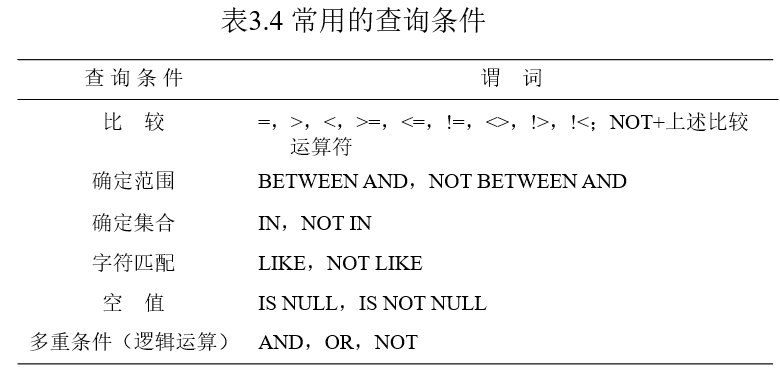
a.消除取值重复的行

如果没有指定DISTINCT关键词，则缺省为ALL

e.g.查询选修了课程的学生学号： SELECT Sno FROM SC；等价于：SELECT ALL Sno FROM SC；

指定DISTINCT关键词，去掉表中重复的行： SELECT DISTINCT Sno FROM SC；

b.查询满足条件的子句



b.1比较大小

e.g.查询计算机科学系全体学生的名单：SELECT Sname FROM Student WHERE Sdept=‘CS’;

查询所有年龄在20岁以下的学生姓名以及年龄：SELECT Sname,Sage FROM Student WHERE Sage < 20;

查询考试成绩有不及格的学生的学号：SELECT DISTINCT Sno FROM SC WHERE GRADE < 60;有一门小于60就可以，一人有多门不及格也只显示一次

b.2确定范围

谓词：BETWEEN...AND...或NOT BETWEEN ...AND...

e.g.查询年龄(不)在[20,23]之间的学生的姓名、系别、年龄：SELECT Sname Sdept,Sage FROM Student WHERE Sage (NOT) BETWEEN 20 AND 23;

b.3确定集合

谓词：IN，NOT IN

e.g.查询(不是)信息系(IS)、数学系(MA)和计算机科学系(CS)学生的姓名和性别：SELECT Sname，Ssex FROM Student WHERE Sdept (NOT) IN ( 'IS'，'MA'，'CS' );

b.4字符匹配

谓词：[NOT]LIKE ''[ESCAPE'']

b.4.1匹配串为固定字符串

e.g.查询学号为200215121的学生的详细情况：SELECT \* FROM Seudent WHERE Sno LIKE '200215121';等价于SELECT \* FROM Seudent WHERE Sno = 200215121;

b.4.2匹配串为含通配符的字符串

%表示任意长度(可以为0)的字符串

\_表示任意的单个字符

e.g.查询所有(不)姓刘学生的姓名：SELECT Sname FROM Student WHERE Sname (NOT) LIKE '刘%'；

查询姓“欧阳“且全名为三个汉字的学生的姓名：SELECT Sname FROM Student WHERE Sname LIKE '欧阳\_'；

查询名字中第二个字为“阳”的学生的姓名：SELECT Sname FROM Student WHERE Sname LIKE '\_阳\_';

b.4.3

使用换码字符将通配符转移为普通字符

e.g.查询DB\_DESIGN课程的课程号和学分：SELECT Cno,Credit FROM Course WHERE Cname LIKE 'DB\-DESIGN'ESCAPE'\' /\*表示跟在DB后面的\_被\换码，不在表示通配符，仅表示\_\*/

查询以"DB\_"开头，且倒数第3个字符为 i的课程的详细情况：SELECT \* FROM Course WHERE Cname LIKE 'DB\\_%i\_ \_' ESCAPE ' \ ‘；/\*第一个\_前面有\所以被转义为普通\_而后面的两个\_前面没有\仍然作为通配符\*/

b.5涉及空值的查询

谓词：IS NULL或IS NOT NULL 注意：IS不能用=代替

e.g.某些学生选修课程后没有参加考试，所以有选课记录，但没有考试成绩。查询没有(有)成绩的学生的学号和相应的课程号：SELECT Sno，Cno FROM SC WHERE Grade IS (NOT) NULL

b.6多重条件查询

逻辑运算符：AND和OR来联结多个查询条件；AND的优先级高于OR，可以用括号改变优先级

可以用来实现多种其他谓词：[NOT]IN [NOT]BTEWEEN...AND...

e.g.查询计算机系年龄小于20的学生的姓名：SELECT Sname FROM Student WHERE Sdept = 'CS' AND Sage<20

改写b.3中的例子：查询信息系(IS)、数学系(MA)和计算机科学系(CS)学生的姓名和性别：SELECT Sname，Ssex FROM  Student WHERE Sdept= 'IS ' OR Sdept= 'MA' OR Sdept= 'CS '；

(3)ORDER BY子句

可以按一个或多个属性列排序。升序：ASC，降序：DESC，缺省默认为升序

当排序列含有空值时：ASC将空值元组最后显示，DESC将空值元组最先显示(将空值视为最大值)

e.g.查询选修了3号课程的学生的学号及其成绩，查询结果按分数降序排列：SELECTSno，Grade     FROM SC  WHERE Cno= ' 3'    ORDER BY Grade DESC；

查询全体学生情况，查询结果按所在系的系号升序排列，同一系中的学生按年龄降序排列：SELECT \*   FROM  Student   ORDER BY Sdept，Sage DESC；

(4)聚集函数

计数：COUNT([DISTINCT|ALL] \*)或 COUNT([DISTINCT|ALL] )

计算总和：SUM([DISTINCT|ALL] )

计算平均值：AVG([DISTINCT|ALL] )

最大最小值： MAX([DISTINCT|ALL] ) MIN([DISTINCT|ALL] )

e.g.查询学生总人数：  SELECT COUNT(\*)   FROM  Student；

     查询选修了课程的学生人数：SELECT COUNT(DISTINCT Sno)   FROM SC；

     计算1号课程的学生平均成绩：SELECT AVG(Grade)   FROM SC   WHERE Cno= '1‘；

查询选修1号课程的学生最高分数：SELECT MAX(Grade) FROM SC WHER Cno= ‘ 1 ’；

查询学生200215012选修课程的总学分数：SELECT SUM(Ccredit) FROM SC， Course WHER Sno='200215012' AND SC.Cno=Course.Cno;

(5)GROUP BY子句

GROUP BY子句分组：

细化聚集函数的作用对象：未对查询结果分组，聚集函数将作用于整个查询结果；对查询结果分组后，聚集函数将分别作用于每个组；作用对象是查询的中间结果表；按指定的一列或多列值分组，值相等的为一组

e.g.求各个课程号及相应的选课人数：SELECT Cno，COUNT(Sno) FROM SC GROUP BY Cno； /\*该语句对查询结果先按照Cno的值分类之后再对每一组进行COUNT计数\*/

查询选修了3门以上课程的学生学号：SELECT Sno FROM SC GROUP BY Sno HAVING COUNT(\*) >3；  先用GROUP BY按照Sno进行分组，在对每一组用COUNT计数，HAVING短语给出了选择组的条件，只有满足条件(元组个数大于3，此学生选修课程数超过三门)的组才会被选出来

(6)HAVING语句

HAVING短语与WHERE子句的区别在于作用对象不同：WHERE子句作用于基表或视图，从中选择满足条件的元组；HAVING短语作用于组，从中选择满足条件的组。

简单的说只有HAVING的子句能是GROUP BY 之后的聚集函数，同时HAVING 也只能在有GROUP BY 而且有聚集函数时才能使用

e.g.查询平均成绩大于90分的学生学号以及平均成绩

wrong：SELECT Sno,AVG(Grade) FROM Sc WHERE AVG(Grade)>90 GROUP BY Sno

WHERE子句中不能用聚集函数作为条件表达式

right:SELECT Sno,AVG(Grade) FROM Sc GROUP BY Sno HAVING AVG (Grade)>90

2.连接查询

(0)简介

1)连接查询：同时涉及多个表的查询； 连接条件或连接谓词：用来连接两个表的条件；

一般格式：.比较运算符. 或者 .BETWEEN.AND.

连接字段：链接谓词中的列名称；连接条件中的各链接字段类型必须是可比的，但名字不必是相同的

2)嵌套循环法(NESTED-LOOP)

首先在表1中找到第一个元组，然后从头开始扫描表2，逐一查找满足连接件的元组，找到后就将表1中的第一个元组与该元组拼接起来，形成结果表中一个元组。

表2全部查找完后，再找表1中第二个元组，然后再从头开始扫描表2，逐一查找满足连接条件的元组，找到后就将表1中的第二个元组与该元组拼接起来，形成结果表中一个元组。

重复上述操作，直到表1中的全部元组都处理完毕

3)排序合并法(SORT-MERGE)

常用于=连接

首先按连接属性对表1和表2排序，对表1的第一个元组，从头开始扫描表2，顺序查找满足连接条件的元组，找到后就将表1中的第一个元组与该元组拼接起来，形成结果表中一个元组。当遇到表2中第一条大于表1连接字段值的元组时，对表2的查询不再继续

找到表1的第二条元组，然后从刚才的中断点处继续顺序扫描表2，查找满足连接条件的元组，找到后就将表1中的第一个元组与该元组拼接起来，形成结果表中一个元组。直接遇到表2中大于表1连接字段值的元组时，对表2的查询不再继续

重复上述操作，直到表1或表2中的全部元组都处理完毕为止

4)索引连接(INDEX-JOIN)

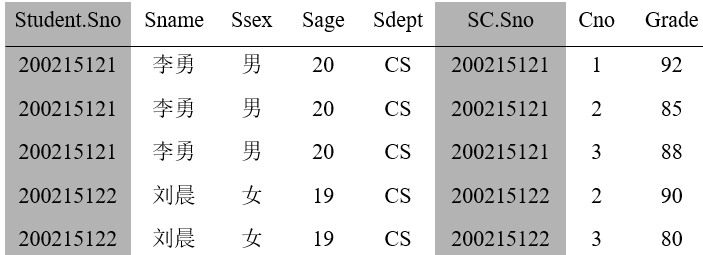
先对表2按连接字段建立索引，对表1中的每个元组，依次根据其连接字段值查询表2的索引，从中找到满足条件的元组，找到后就将表1中的第一个元组与该元组拼接起来，形成结果表中一个元组

(1)等值与非等值连接查询

a.等值连接

等值连接运算符为：=

e.g.查询每个学生及其选修课程的情况：SELECT Student.\* ,SC.\* FROM Student,SC WHERE Student.Sno = Sc.Sno /\*用Sno将Student和SC表连接\*/



b.自然连接：

查询每个学生及其选修课程的情况：SELECT Student.Sno，Sname，Ssex，Sage，Sdept，Cno，Grade FROM Student，SC WHERE Student.Sno = SC.Sno；

(2)自身连接

一个表与自己进行连接；需要给表起别名来加以区别；由于所有属性名都是同名属性，故必须使用别名前缀

e.g.查询每一门课的间接先修课(先修课的先修课)：SELECT FIRST.Cno，SECOND.Cpno FROM Course FIRST，Course SECOND WHERE FIRST.Cpno = SECOND.Cno；/\*默认Cpno不能为空\*/

(3)外连接

外连接与普通连接的区别：普通连接操作只输出满足链接条件的元组；外连接操作以指定表为连接主体，将主题表中不满足连接条件的元组一并输出。有右外连接和左外连接

e.g.查询每个学生及其选修课程的情况SELECT Student.Sno，Sname，Ssex，Sage，Sdept，Cno，Grade FROM Student LEFT OUT JOIN SC ON (Student.Sno = SC.Sno)；

(4)复合条件连接

复合条件连接：WHERE子句中含多个连接条件

e.g.查询选修2号课程且成绩在90分以上的所有学生：  SELECTStudent.Sno, Sname FROM   Student, SC WHERE Student.Sno = SC.Sno AND /\*连接谓词+其他限定条件\*/SC.Cno= ‘2’ AND SC.Grade >90；

查询每个学生的学号、姓名、选修的课程名及成绩：SELECT Student.Sno，Sname，Cname，Grade FROM Student，SC，Course /\*多表连接\*/ WHERE Student.Sno = SC.Sno AND SC.Cno = Course.Cno；

3.嵌套查询

(0)概述

一个SELECT-FROM-WHERE成为一个查询块；将一个查询块嵌套在另一个查询块的WHERE子句或HAVING短语的条件中的查询称为嵌套查询(即为多重循环)

SELECT Sname /\*外层查询/父查询\*/

FROM Student WHERE Sno IN

(SELECT Sno /\*内层查询/子查询\*/

FROM SC

WHERE Cno= ' 2 ')；

子查询的限制：不能使用ORDER BY子句；层层嵌套的方式反映了SQL语言的结构化；有些嵌套查询可以用连接运算替代

不相关子查询：子查询的查询条件不依赖与父查询，由里向外，逐层处理。每个子查询在上一级查询处理之前求解，子查询的结果用于建立其父查询的查询条件

相关子查询：子查询的查询条件依赖于父查询：首先取外层查询中表的第一个元组，根据它与内层查询相关的属性值处理内层查询，若WHERE子句返回值为真，则取此元组放入结果表；然后再取外层表的下一个元组；重复这一过程，直至外层表全部检查完为止

(1)带有IN谓词的子查询

e.g.查询与“刘晨”在同一个系学习的学生：

先确定“刘晨”所在系名 再查找所有在IS系学习的学生。

SELECT Sdept SELECT Sno，Sname，Sdept

FROM Student FROM Student

WHERE Sname= ' 刘晨 '； WHERE Sdept= ' CS '；

结果为： CS

将第一步查询嵌入到第二步查询的条件中(此查询为不相关子查询)

SELECT Sno，Sname，Sdept

FROM Student

WHERE Sdept IN

(SELECT Sdept

FROM Student

WHERE Sname= ‘ 刘晨 ’)；

此查询可以用自身连接查询完成

SELECT S1.Sno，S1.Sname，S1.Sdept

FROM Student S1，Student S2

WHERE S1.Sdept = S2.Sdept AND S2.Sname = '刘晨'；

查询选修了课程名为“信息系统”的学生学号和姓名

SELECT Sno，Sname ③ 最后在Student关系中

FROM Student 取出Sno和Sname

WHERE Sno IN

(SELECT Sno ② 然后在SC关系中找出选

FROM SC 修了3号课程的学生学号

WHERE Cno IN

(SELECT Cno ① 首先在Course关系中找出

FROM Course “信息系统”的课程号，为3号

WHERE Cname= ‘信息系统’));

用连接查询实现

SELECT Sno，Sname

FROM Student，SC，Course

WHERE Student.Sno = SC.Sno AND SC.Cno = Course.Cno AND Course.Cname=‘信息系统’；

(2)带有比较运算符的子查询

当能确切知道内层查询返回单值时，可以用比较运算符(>,=,<=,!=或<>)

与ANY或ALL谓词配合使用

假设一个学生只可能在一个系学习，并且必须属于一个系，则查询与“刘晨”在同一个系学习的学生的嵌套查询中可以用 = 代替IN ：

SELECT Sno，Sname，Sdept

FROM Student

WHERE Sdept =

(SELECT Sdept

FROM Student

WHERE Sname= ‘刘晨’)；

子查询一定要缝在比较符之后

错误例子：SELECT Sno，Sname，Sdep FROM Student

WHERE ( SELECT Sdept

FROM Student

WHERE Sname= ‘ 刘晨 ’ )

= Sdept；

带有比较运算符的子查询

e.g.找出每个学生超过他选修课程平均成绩的课程号：x是SC的一个别名，又称为元组变量，可以用来不表示SC的一个元组，

SELECT Sno， Cno

FROM SC x

WHERE Grade >=(SELECT AVG(Grade)

FROM SC y

WHERE y.Sno=x.Sno);

该查询的一个可能的执行过程：

从外层查询中取出SC的一个元组x，将元组x的Sno值(200215121)传送给内层查询。

SELECT AVG(Grade) FROM SC y WHERE y.Sno='200215121';

执行内层查询，得到值88(近似值)，用该值代替内层查询，得到外层查询：

SELECT Sno， Cno FROM SC x WHERE Grade >=88；

(3)带有ANY(SOME)或ALL谓词的子查询

谓词语义：ANY任意一个值；ALL所有值

> ANY 大于子查询结果中的某个值

> ALL 大于子查询结果中的所有值

< ANY 小于子查询结果中的某个值

< ALL 小于子查询结果中的所有值

>= ANY 大于等于子查询结果中的某个值

>= ALL 大于等于子查询结果中的所有值

<= ANY 小于等于子查询结果中的某个值

<= ALL 小于等于子查询结果中的所有值

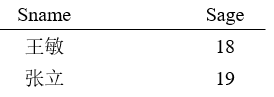
= ANY 等于子查询结果中的某个值

=ALL 等于子查询结果中的所有值(通常没有实际意义)

!=(或<>)ANY 不等于子查询结果中的某个值

!=(或<>)ALL 不等于子查询结果中的任何一个值

e.g.查询其他系中比计算机科学任意一学生年龄小的学生姓名和年龄



用ANY谓词：SELECT Sname，Sage FROM Student WHERE Sage < ANY (SELECT Sage FROM Student WHERE Sdept= ' CS ') AND Sdept <> ‘CS ' ; /\*父查询块中的条件 \*/

用聚集函数实现：SELECT Sname，Sage FROM Student WHERE Sage < (SELECT MAX(Sage) FROM Student WHERE Sdept= ‘CS ') AND Sdept <> ' CS ’;

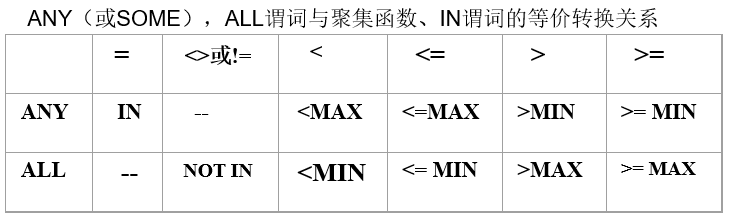
RDBMS执行此查询时，首先处理子查询，找出CS系中所有学生的年龄，构成一个集合(20，19)之后处理父查询，找所有不是CS系且年龄小于20 或 19的学生

e.g.查询其他系中比计算机科学系所有学生年龄都小的学生姓名及年龄

0

用ALL谓词: SELECT Sname，Sage FROM Student WHERE Sage < ALL (SELECT Sage FROM Student WHERE Sdept= ' CS ') AND Sdept <> ' CS ’;

用聚集函数：SELECT Sname，Sage FROM Student WHERE Sage < (SELECT MIN(Sage) FROM Student WHERE Sdept= ' CS ') AND Sdept <>' CS ’;



(4)带有EXISTS谓词的子查询($表示存在#表示任意->表示蕴含~表示非)

EXISTS谓词：存在量词$；带有EXISTS谓词的子查询不返回任何数据，只产生逻辑真值“true”或逻辑假值“false”；若内层查询结果非空，则外层的WHERE子句返回真值；若内层查询结果为空，则外层的WHERE子句返回假值 ；由EXISTS引出的子查询，其目标列表达式通常都用\* ，因为带EXISTS的子查询只返回真值或假值，给出列名无实际意义

NOT EXISTS谓词：若内层查询结果非空，则外层的WHERE子句返回假值；若内层查询结果为空，则外层的WHERE子句返回真值

e.g.查询所有(没)选修了1号课程的学生姓名。

用嵌套查询：SELECT Sname FROM Student WHERE (NOT) EXISTS (SELECT \* FROM SC WHERE Sno=Student.Sno AND Cno= ' 1 ')；

用连接运算：SELECT Sname FROM Student, SC WHERE Student.Sno=SC.Sno AND SC.Cno != '1';

不同形式的查询间的替换：一些带EXISTS或NOT EXISTS谓词的子查询不能被其他形式的子查询等价替换；所有带IN谓词、比较运算符、ANY和ALL谓词的子查询都能用带EXISTS谓词的子查询等价替换

用EXISTS/NOT EXISTS实现全称量词(难点)：SQL语言中没有全称量词#(For all)；可以把带有全称量词的谓词转换为等价的带有存在量词的谓词：(#x)P ≡ ~($x(~P))

e.g.查询与“刘晨”在同一个系学习的学生：用带EXISTS谓词的子查询替换：

SELECT Sno，Sname，Sdept

FROM Student S1 WHERE EXISTS

(SELECT \* FROM Student S2

WHERE S2.Sdept = S1.Sdept AND S2.Sname = ‘刘晨’)；

e.g.查询选修了全部课程的学生姓名：没有一门课是他不修的

SELECT Sname FROM Student

WHERE NOT EXISTS

(SELECT \* FROM Course

WHERE NOT EXISTS

(SELECT \* FROM SC

WHERE Sno= Student.Sno AND Cno= Course.Cno))；

用EXISTS/NOT EXISTS实现逻辑蕴函(难点)：SQL语言中没有蕴函(Implication)逻辑运算；可以利用谓词演算将逻辑蕴函谓词等价转换为：p -> q ≡ ~p∨q

e.g.查询至少选修了学生200215122选修的全部课程的学生号码

用逻辑蕴函表达：查询学号为x的学生，对所有的课程y，只要200215122学生选修了课程y，则x也选修了y。形式化表示：用P表示谓词 “学生200215122选修了课程y”，用q表示谓词 “学生x选修了课程y”，则上述查询为: (#y) p->q

等价变换：(#y)p->q ≡ ~($y (~(p -> q )) ≡ ~($y (~(~ p∨ q) )) ≡ ~ $y(p∧~q) 变换后语义：不存在这样的课程y，学生200215122选修了y，而学生x没有选。

用NOT EXISTS谓词表示：

SELECT DISTINCT Sno

FROM SC SCX

WHERE NOT EXISTS

(SELECT \*

FROM SC SCY

WHERE SCY.Sno = ' 200215122 ' AND

NOT EXISTS

(SELECT \*

FROM SC SCZ

WHERE SCZ.Sno=SCX.Sno AND

SCZ.Cno=SCY.Cno))；

4.集合查询

集合操作的种类：并操作(UNION)；交操作(INTERSECT)；差操作(EXCEPT)

参加集合操作的各查询结果的列数必须相同；对应项的数据类型也必须相同

(1)并(UNION)

UNION：将多个查询结果合并起来时，系统自动去掉重复元组。

UNION ALL：将多个查询结果合并起来时，保留重复元组

e.g.查询计算机科学系的学生及年龄不大于19岁的学生。

SELECT \* SELECT DISTINCT \*

FROM Student FROM Student

WHERE Sdept= 'CS' WHERE Sdept= 'CS' OR Sage<=19；

UNION

SELECT \*

FROM Student

WHERE Sage<=19；

e.g.查询选修了课程1或者选修了课程2的学生。

SELECT Sno SELECT Sno

FROM SC From SC

WHERE Cno=' 1 ' WHERE Cno = '1' OR Cno = '2';

UNION

SELECT Sno

FROM SC

WHERE Cno= ' 2 '；

(2)交(INEERSECT)

e.g.查询计算机科学系的学生与年龄不大于19岁的学生的交集

SELECT \* SELECT \*

FROM Student FROM Student

WHERE Sdept='CS' WHERE Sdept= 'CS' AND Sage<=19；

INTERSECT

SELECT \*

FROM Student

WHERE Sage<=19

e.g.查询选修课程1的学生集合与选修课程2的学生集合的交集

SELECT Sno SELECT Sno

FROM SC FROM SC

WHERE Cno=' 1 ' WHERE Cno=' 1 ' AND Sno IN

INTERSECT (SELECT Sno

SELECT Sno FROM SC

FROM SC WHERE Cno=' 2 ')；

WHERE Cno='2 '；

(3)差(EXCEPT)

e.g.查询计算机科学系的学生与年龄不大于19岁的学生的差集。

SELECT \* SELECT \*

FROM Student FROM Student

WHERE Sdept='CS' WHERE Sdept= 'CS' AND Sage>19；

EXCEPT

SELECT \*

FROM Student

WHERE Sage <=19;

(五)数据更新

1.插入数据

两种插入数据的方式：插入元组；插入子查询结果-->可以一次插入多个元组

(1)插入元组

格式：INSERT

INTO [(，...)]

VALUES (,...)

功能：将新元组插入指定表中

INTO子句：属性列的顺序可以和表定义中的顺序不一致；没有指定属性列；指定部分属性列

VALUES子句：提供的值必须与INTO子句匹配-->值的个数+值的类型

e.g.将一个新学生元组(学号：200215128；姓名：陈冬；性别：男；所在系：IS；年龄：18岁)插入到Student表中。

INSERT

INTO Student (Sno，Sname，Ssex，Sdept，Sage)

VALUES ('200215128'，'陈冬'，'男'，'IS'，18)；

将学生张成民的信息插入到Student表中。

INSERT

INTO Student

VALUES (‘200215126’， ‘张成民’， ‘男’，18，'CS');

插入一条选课记录( '200215128'，'1 ')。

INSERT

INTO SC(Sno，Cno)

VALUES (‘ 200215128 ’，‘ 1 ’)； /\*RDBMS将在新插入记录的Grade列上自动地赋空值\*/

或者：

INSERT

INTO SC

VALUES (' 200215128 '，' 1 '，NULL)；

(2)插入子查询结果

格式：INSERT

INTO [(，...)]

子查询；

功能：将子查询结果插入指定表中

INTO子句：与插入元组类似

子查询：SELECT子句目标必须与INTO子句匹配-->值的个数+值的类型

e.g.对每一个系，求学生的平均年龄，并把结果存入数据库。

第一步：建表

CREATE TABLE Dept\_age

(Sdept CHAR(15) /\* 系名\*/

Avg\_age SMALLINT)； /\*学生平均年龄\*/

第二步：插入数据

INSERT INTO Dept\_age(Sdept，Avg\_age)

SELECT Sdept，AVG(Sage)

FROM Student

GROUP BY Sdept；

注：RDMS在执行插入语句时会检查所插元组是否破坏了表上已经定义的完整性规则：实体完整性；参照完整性；用户定义的完整性(NOT NULL约束/UNIQUE/值域约束)

2.修改数据

语句格式:

UPDATE

SET=[=]...

WHERE；

功能：修改指定表中满足WHERE子句条件的元组

SET子句：指定修改方式；要修改的列；修改后的值

WHERE子句：指定要修改的元组；缺省表示要修改表中的所有元组

三种修改方式：修改某一个元组的值；修改多个元组的值；带子查询的修改语句

(1)修改某一个元组的值

e.g.将学生200215121的年龄改为22岁

UPDATE Student

SET Sage=22

WHERE Sno=' 200215121 '；

(2)修改多个元组的值

e.g.将所有学生的年龄增加1岁

UPDATE Student

SET Sage= Sage+1；

(3)带子查询的修改语句

e.g.将计算机科学系全体学生的成绩置零。

UPDATE SC

SET Grade=0

WHERE 'CS'=

(SELETE Sdept

FROM Student

WHERE Student.Sno = SC.Sno)；

注：RDMS在执行修改语句时会检查所插元组是否破坏了表上已经定义的完整性规则：实体完整性；参照完整性；用户定义的完整性(NOT NULL约束/UNIQUE/值域约束)

3.删除数据

语句格式：

DELETE

FROM

WHERE

功能：删除指定表中满足WHERE子句条件的元组

WHERE子句：指定要删除的元组，缺省表示要删除表中的全部元组，表的定义仍在字典中

三种删除方式：删除某一个元组的值；删除多个元组的值；带子查询的删除语句

(1)删除某一个元组的值

DELETE

FROM Student

WHERE Sno = 200215128；

(2)删除多个元组的值

DELETE

FROM SC；

使SC变成空表

(3)带子查询的删除语句

DELETE FROM SC

WHERE ‘CS’=

(SELECT Sdept

FROM Student

WHERE Student。Sno=SC.Sno);

(六)空值的处理

1.空值定义及情况

所谓空值就是“不知道”，“不存在”，“无意义”的值，是一个很特殊的值，含有不确定性。SQL允许某些选组的某些属性在一定情况下取空值：

(1)该属性应该有一个值，但目前不知道它的具体值，如学生的年龄属性

(2)该属性不应该有值，如学生缺考的成绩

(3)由于某种原因不便于填写，如个人的手机号码

(4)因为外连接而产生的空值

2.空值的判断

用IS NULL或IS NOT NULL判断，不能用==或者!=,<>

3.空值的约束条件

属性定义或者域定义中有NOT NULL约束条件的不能取空值，码属性不能取空值

4.空值的算数运算、比较运算、逻辑运算

空值与所有值的算数运算结果为空值

空值与了所有值的逻辑运算为UNKNOWN

T:TRUE F:FALSE U:UNKNOWN

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| x | y | x AND y | x OR y | NOT x |
| T | T | T | T | F |
| T | U | U | T | F |
| T | F | F | T | F |
| U | T | U | T | U |
| U | U | U | U | U |
| U | F | F | U | U |
| F | T | F | T | T |
| F | U | F | U | T |
| F | F | F | F | T |

(七)视图

视图是虚表，是从一个或几个基本表(或视图)导出的表；只存放视图的定义，不存放视图对应的数据；基表中数据发生变化，从视图中查询出的数据也会随之改变

基于视图的操作：查询；删除；受限更新；定义基于该视图的新视图

1.定义视图

(1)建立视图

1)语句格式：CREATE VIEW<>[(,...)]

AS 子查询

[WITH CHECK ORTION]

组成视图的属性列名；全部省略或者全部指定；子查询中不允许含有ORDER BY子句和DISTINCT短语

RDBMS执行CREATE VIEW语句时只是把视图定义存入数据字典，并不执行其中的SELECT语句；在对视图查询时，按视图的定义从基本表中将数据查出

e.g.建立信息系学生的视图：

CREATE VIEW IS\_Student

AS

SELECT Sno，Sname，Sage

FROM Student

WHERE Sdept= 'IS'

建立信息系学生的视图，并要求进行修改和插入操作时仍需保证该视图只有信息系的学生 。

CREATE VIEW IS\_Student

AS

SELECT Sno，Sname，Sage

FROM Student

WHERE Sdept= 'IS'

WITH CHECK OPTION；

对IS\_Student视图的更新操作：

修改、删除操作：自动加上Sdept=‘IS’的条件；插入操作，自动检查Sdept属性值是否为‘IS’，如果不是，拒绝该插入操作；如果没有提供Sdept属性值，自动定义Sdept为‘IS’

2)基于多个基表的视图

e.g.建立信息系选修了1号课程的学生视图。

CREATE VIEW IS\_S1(Sno，Sname，Grade)

AS

SELECT Student.Sno，Sname，Grade

FROM Student，SC

WHERE Sdept= 'IS' AND Student.Sno=SC.Sno AND SC.Cno= '1'；

3)基于视图的视图

e.g.建立信息系选修了1号课程且成绩在90分以上的学生的视图。

CREATE VIEW IS\_S2

AS

SELECT Sno，Sname，Grade

FROM IS\_S1

WHERE Grade>=90；

4)带表达式的视图

e.g.定义一个反映学生出生年份的视图。

CREATE VIEW BT\_S(Sno，Sname，Sbirth)

AS

SELECT Sno，Sname，2000-Sage

FROM Student；

5)分组视图

e.g.将学生的学号及他的平均成绩定义为一个视图，假设SC表中“成绩”列Grade为数字型

CREAT VIEW S\_G(Sno，Gavg)

AS

SELECT Sno，AVG(Grade)

FROM SC

GROUP BY Sno；

6)不指定属性列

e.g.将Student表中所有女生记录定义为一个视图

CREATE VIEW F\_Student(F\_Sno，name，sex，age，dept)

AS

SELECT \*

FROM Student

WHERE Ssex=‘女’；

缺点：修改基表Student的结构后，Student表与F\_Student视图的映象关系被破坏，导致该视图不能正确工作。

(2)删除视图

语句格式：DPOP VIEW

该语句从数据字典中删除指定的视图定义；如果该是图上还导出了其他视图，使用CASCADE级联删除语句，该视图和由它导出的所有视图一起删除；删除基表时，由该基表导出的所有视图定义都必须显式地使用DROP VIEW语句删除

e.g. 删除视图BT\_S： DROP VIEW BT\_S；

删除视图IS\_S1：DROP VIEW IS\_S1； 拒绝执行

级联删除：DROP VIEW IS\_S1 CASCADE;

2.查询视图

用户角度：查询视图与查询基本表相同

RDBMS实现视图查询的方法：视图消解法：进行有效性检查；转换成等价的对基本表的查询；执行修正后的查询

e.g.在信息系学生的视图中找出年龄小于20岁的学生。

SELECT Sno，Sage

FROM IS\_Student

WHERE Sage

IS\_Student视图的定义 (参见视图定义例1)

视图消解转换后的查询语句：

SELECT Sno，Sage

FROM Student

WHERE Sdept= 'IS' AND Sage

e.g.查询选修了1号课程的信息系学生：

SELECT IS\_Student.Sno，Sname

FROM IS\_Student，SC

WHERE IS\_Student.Sno =SC.Sno AND SC.Cno= '1'；

视图消解法的局限：有些情况下，视图消解法不能生成正确的查询

e.g.在S\_G视图中查询平均成绩在90分以上的学生学号和平均成绩

SELECT \*

FROM S\_G

WHERE Gavg>=90；

S\_G视图的子查询定义：

CREATE VIEW S\_G (Sno，Gavg)

AS

SELECT Sno，AVG(Grade)

FROM SC

GROUP BY Sno；

查询转换：

wrong：SELECT Sno，AVG(Grade)

FROM SC

WHERE AVG(Grade)>=90

GROUP BY Sno；

right：SELECT Sno，AVG(Grade)

FROM SC

GROUP BY Sno

HAVING AVG(Grade)>=90；

3.更新视图

1)更新

e.g.将信息系学生视图IS\_Student中学号200215122的学生姓名改为“刘辰”。

UPDATE IS\_Student

SET Sname= '刘辰'

WHERE Sno= ' 200215122 '；

转换后的语句：

UPDATE Student

SET Sname= '刘辰'

WHERE Sno= ' 200215122 ' AND Sdept= 'IS'；

2)插入

e.g.向信息系学生视图IS\_S中插入一个新的学生记录：200215129，赵新，20岁

INSERT

INTO IS\_Student

VALUES(‘95029’，‘赵新’，20)；

转换为对基本表的更新：

INSERT

INTO  Student(Sno，Sname，Sage，Sdept)

VALUES(‘200215129 '，'赵新'，20，'IS' )；

3)删除

e.g.删除信息系学生视图IS\_Student中学号为200215129的记录

DELETE

FROM IS\_Student

WHERESno= ' 200215129 '；

转换为对基本表的更新：

DELETE

FROM Student

WHERESno= ' 200215129 ' AND Sdept= 'IS'；

4)局限

更新视图的限制：一些视图是不可以更新的，因为对这些视图的更新不能唯一地有意义地转换成对相应基本表的更新

e.g.视图S\_G为不可更新视图。

UPDATE  S\_G

SET    Gavg=90

WHERE  Sno= ‘200215121’；

这个对视图的更新无法转换为对基本表SC的更新

更新视图：允许行列子集视图进行更新；对其他类型视图的更新不同系统有不同限制

4.视图的5点作用

简化用户的操作；使用户能以多种角度看待同一数据；对重构数据库提供了一定程度的逻辑独立性；能够对机密数据提供安全保护；适当的利用视图可以更清晰的表达查询

四、数据库安全性

问题提出：数据库的一大特点是数据可以共享；数据共享必然带来数据库的安全性问题；数据库系统中的数据共享不能是无条件的共享。

数据库安全性：保护数据库以防止不合法的使用所造成的数据泄露、更改或破坏

1.计算机安全性概述

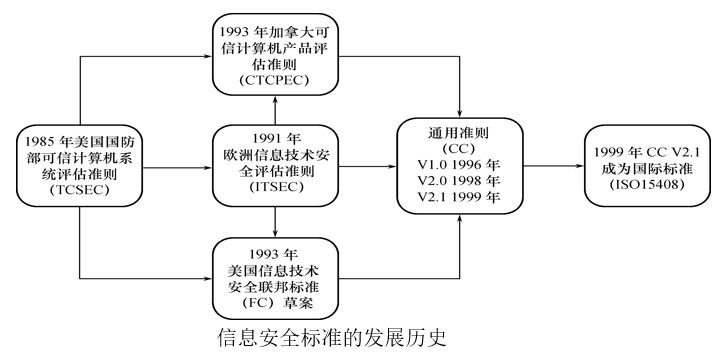
(1)计算机系统的三类安全性问题

计算机系统安全性：为计算机系统建立和采取的各种安全保护措施，以保护计算机系统中的硬件、软件以及数据，防止其因偶然或恶意的原因使系统遭到破坏，数据遭到更改或泄露

三类安全性问题为：技术安全类；管理安全类；政策法律类

(2)安全标准简介

TCSEC标准 CC标准



TCSEC/TDL标准的基本内容：从四个方面来描述安全性级别划分的指标：安全策略；责任；保证；文档

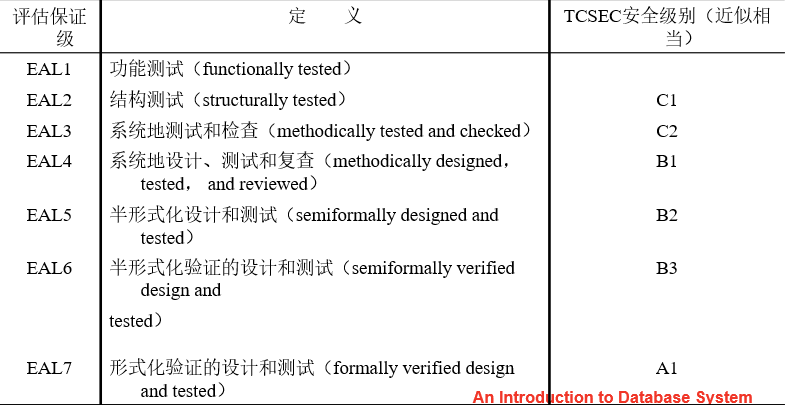
TESEC/TDL安全级别划分：B1为MAC，C1为DAC。该图按系统可靠或可信度信度逐渐增高；各安全级别之间偏序向下兼容；B2以上的系统目前还处于理论研究阶段，应用于一些特殊的部门(军事等)，美国正在大力发展安全产品，试图将目前仅限于少数领域应用的B2安全级别下放到商业应用中来，并逐步成为新的商业标准



CC：提出国际公然的表述信息技术安全性的结构；把信息产品的安全要求分为安全功能要求和安全保证要求

CC文本组成：简介和一般模型；安全功能要求；安全保证要求

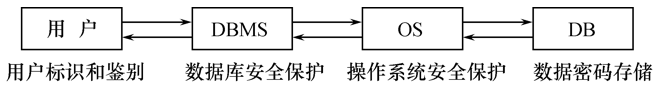
CC评估保证级别划分：



2.数据库安全性控制

非法使用数据库的情况：编写合法程序绕过DBMS及其授权机制；直接或编写应用程序执行非授权操作；通过多次合法查询数据库从中推导出一些保密数据

计算机系统中，安全措施是一级一级层层设置的(下图为计算机系统的安全模型)



数据库安全性控制的常用方法：用户标识和鉴定；存取控制；视图；审计；密码存储；

(1)用户标识和鉴别(Identification & Authentication)

系统提供的最外层安全保护措施

用户标识：有用户名和用户标识号组成(UID)

口令：系统核对口令以鉴别用户身份，静态口令鉴别(密码)，动态口令鉴别(验证码)

用户名和口令易被窃取：每个用户预先约定好一个计算过程或函数

生物鉴别：指纹；人脸；瞳孔等

智能卡鉴别：一种不可复制的硬件，从智能卡中读取数据是静态的，数据会被截获，有安全隐患，实际中采用个人身份识别码(PIN)和智能卡相结合的方式

(2)存取控制

存取控制机制组成：定义用户权限；合法权限检查

用户权限定义和合法权检查机制一起组成了DBMS的安全子系统

常用的存取控制方法：自主存取控制(DAC)：C1级别，灵活；强制存取控制(MAC)：B1级别，严格

在自存取控制方法中，用户对于不同的数据库对象有不同的存取权限，不同的用户对同一对象也有不同的权限，而且用户还可以将其拥有的存取权限转授给其他用户，因此，自存取控制比较灵活

在强制存取控制方法中，每一个数据库对象被标以一定的密集，每一个用户被授予一个级别的许可证。对任意一个对象，只有具有合法许可证的用户才可以存取。强制存取控制因此相对较为严格

(3)自主存取控制方法(DAC-->C1)

可能存在数据的“无意泄露”

原因：这种机制仅仅通过对数据的存取权限来进行安全控制，而数据本身并无安全性标记

解决：对系统控制下的所有主客体实施强制存取策略

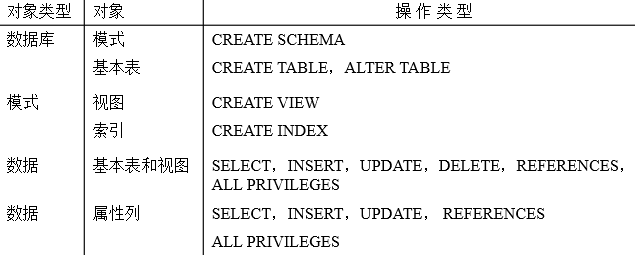
通过SQL的GRANT语句和REVOKE语句实现

用户权限组成：数据对象；操作类型

定义用户存取权限：定义用户可以在那些数据库对象上进行哪些类型的操作

定义存取权限称为授权

关系数据库系统中存取控制对象：(下图为关系数据库中的存取权限)



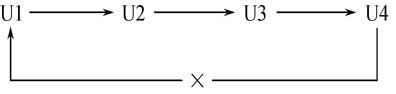
(4)授权与回收

1)GRANT

GRANT语句一般格式：GRANT[，...] ON [] TO[,] [WITH GRANT OPTION];

语义：将对指定操作对象的指定操作权限授予指定的用户

发出GRANT：DBA；数据库对象的创建者(属主Owner)；拥有该权限的用户



接受权限的用户：一个或多个用户；PUBLIC(全体用户)

WITH GRANT OPTION子句：指定后用户可以再次传播

不允许循环授权(如右图)

e.g.把查询Student表权限授给所有用户：GRANT SELECT ON TABLE Student TO PUBLIC;

e.g.把对Student表和Course表的全部权限授予用户U2和U3：GRANT ALL PRIVILIGES ON TABLE Student, Course TO U2, U3;

e.g.把查询Student表和修改学生学号的权限授给用户U4，并允许他再将此权限授予其他用户 ：GRANT UPDATE(Sno), SELECT ON TABLE Student TO U4 WITH GRANT OPTION

对属性列的授权时必须明确指出相应属性列名

2)REVOKE

授予的权限可以由DBA或其他授权者用REVOKE语句收回

REVOKE语句一般格式：REVOKE [,...] [ON] FROM[,...];

e.g.把所有用户修改学生学号的权限收回：

REVOKE UPDATE(Sno) ON TABLE Student FROM PUBLIC；

e.g.把用户U5对SC表的INSERT权限收回：

REVOKE INSERT ON TABLE SC FROM U5 CASCADE ;

将用户U5的INSERT权限收回的时候必须级联(CASCADE)收回；系统只收回直接或间接从U5处获得的权限

3)小结

DBA拥有所有对象的所有权限，可以将不同的权限授予不同的用户

用户：拥有自己建立的对象的全部的操作权限，GRANT-->将自己拥有的权限授予其他用户

被授权的用户：如有WITH GRANT OPTION，则可以再次授予他人

所有授予出去的权限在必要时都可以用REVOKE语句收回

4)创建数据库模式的权限

DBA在创建用户时实现

CREATE USER语句格式：CREATE USER

[WITH][DBA | RESOURCE | CONNECT]

权限与可执行的操作对照表：



(5)数据库角色(ROLE)

数据库角色：被命名的一组与数据库操作相关的权限：角色是权限的集合；可以为一组具有相同权限的用户创建一个角色；简化了授权的过程

角色的创建：CREATE ROLE

给角色授权：GRANT [,...] ON [....] TO [,...]

将一个角色授予其他的角色或用户：GRANT [,...] TO [,...] [WITH GRANT OPTION]

角色权限的收回：REVOKE [,...] ON FROM [,...]

e.g.通过角色来实现将一组权限授予一个用户的步骤如下：

首先创建一个角色 R1 ：CREATE ROLE R1；

然后使用GRANT语句，使角色R1拥有Student表的SELECT、UPDATE、INSERT权限：GRANT SELECT，UPDATE，INSERT ON TABLE Student TO R1；

将这个角色授予王平，张明，赵玲。使他们具有角色R1所包含的全部权限：GRANT R1 TO 王平，张明，赵玲；

可以一次性通过R1来回收王平的这3个权限：REVOKE R1 FROM 王平；

角色的权限修改

GRANT DELETE

ON TABLE Student

TO R1

角色权限收回

REVOKE SELECT

ON TABLE Student

FROM R1；

(6)强制存取控制方法(MAC-->B1)

MAC：保证更高程度的安全性；用户不能直接感知或者进行控制；适用于对数据有严格而固定密级分类的部门(如军事、政府等)

主体是系统中的活动实体：DBMS所管理的实际用户；代表用户的各进程

客体是系统中的被动实体，是受主题操纵的：文件；基表；索引；视图

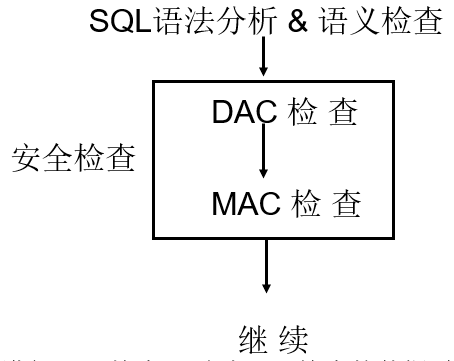
敏感度标记：绝密(Top Secret)(TS) >= 机密(Secret)(S) >= 可信(Confidential)(C) >= 公开(Public)(P)

主体的敏感度标记称为许可证级别，客体的敏感度标记称为密级

仅当主题的许可证级别大于等于客体的密级时，该主体才能读取相应的客体；仅当主体的许可证级别小于等于客体的密级时，该主体才能写相应的客体；

修正规则：主体的许可证级别<=客体的密级-->主体能写客体

规则的共同点：禁止了拥有高许可证级别的主题更新低密级的数据对象，使得更低许可证级别的人对其有存取权限

DAC和MAC共同构成DBMS的安全机制；实现MAC时首先要实现DAC(较高安全性级别提供的安全性保护要包含较低级别的所有保护)；

DAC+MAC安全检查示意图：

先进性DAC检查，通过DAC检查的数据对象在由系统进行MAC检查，只有通过MAC检查的数据对象才能进行存取

3.视图机制

把要保密的数据对无权存取这些数据的用户隐藏起来，对数据提供一定程度的安全保护：主要功能是提供数据独立性，无法完全满足要求；间接实现了支持存取谓词的用户权限定义

e.g.建立计算机系学生的视图，把对该视图的SELECT权限授于王平，把该视图上的所有操作权限授于张明

先建立计算机系学生的视图：CS\_Student CREATE VIEW CS\_Student AS SELECT \* FROM Student WHERE Sdept='CS'；

在视图上进一步定义存取权限：GRANT SELECT ON CS\_Student TO 王平; GRANT ALL PRIVILIGES ON CS\_Student TO 张明

4.审计(Audit)

(0)概述

审计的功能：审计日志-->将用户对数据库的所有操作记录在上面；DBA利用审计日志-->找出非法存取数据的人、时间、内容；C2以上安全级别的DBMS必须具有

基本功能，提供多种审计查阅方式：基本的、可选的、有限的等

提供多套审计规则：审计规则一般在数据库初始化时设定

提供审计分析和报表功能

审计日志管理功能，包括为防止审计员误删审计记录，审计日志必须先转储后删除，对转储的审计记录文件提供完整性和保密性保护，只允许审计员查阅和转储审计记录，不允许任何用户新增和修改审计记录

系统提供查询审计设置以及审计记录信息的专门视图，对于系统权限级别、语句界别及模式对象级别的审计记录也可以通过相关的系统表直接查看

(1)分类

1)用户级审计

针对自己创建的数据库表或视图进行审计；记录所有用户对这些表或视图的一切成功和(或)不成功的访问要求以及各种类型的SQL操作

2)系统集审计

DBA设置；监测成功或失败的登录要求；监测GRANT和REVOKE操作以及其他数据库级权限下的操作

(2)审计事件(在书上)

服务器事件：审计数据库服务器发生的事件，包括数据库服务器的启东、停止、数据库服务器配置文件的重新加载

系统权限：对系统拥有的结构或模式对象进行操作的审计，要求该操作的权限是通过西戎权限获得的

语句事件：对SQL语句，如DDL、DML、DQL(Data Query Language，数据查询语言)、以及DCL语句的审计

模式对象事件：对特定模式对象上进行的SELECT或DML操作的审计。模式第一项包括表、视图、存储过程、函数等。模式对象不包括依附于表的索引、约束、触发器、分区表等

(2)AUDIT和NOAUDIT

AUDIT：设置审计功能 NOAUDIT：取消审计功能

e.g.对修改SC表结构或修改SC表数据的操作进行审计：

AUDIT ALTER，UPDATE

ON SC；

取消对SC表的一切审计

NOAUDIT ALTER，UPDATE

ON SC；

5.数据加密

数据加密是为了防止数据在村塾恶核传输中失密的有效手段；

加密的基本思想是根据一定的算法将原始数据--明文变换为不可以直接识别的格式--密文，从而使得不知道解密算法的人无法货值数据的内容

加密方法：替换方法；置换方法；混合方法

数据加密主要包括存储加密和传输加密存储加密方式。

1)存储加密

一般提供透明和非透明两种，存储加密是内核级加密保护方式，对用户完全透明；非透明存储加密是通过多个加密函数实现的。透明存储加密是数据在写到磁盘时对数据进行加密，授权用户读取数据时在对其进行解密。由于数据加密对用户透明，数据库的应用程序不需要做任何修改，至于要在创建表语句中说明余姚加密的字段即可。当对加密数据进行增删改查询操作时，数据库关系系统自动对数据进行加密解密公国。基于数据库内核的数据存储加密、解密方法性能较好，安全完备性较高

2)传输加密

在客户/服务器结构中，数据库用户与服务器之间如果采用明文的方式传输数据，容易被网络恶意用户截获、篡改，存在安全樱花，因此，为保证二者之间的安全数据交换，数据库管理系统提供了传输加密的功能常用的传输加密方式有链路加密和端到端加密(具体看书)

6.统计数据库安全性

统计数据库：允许数据查询聚集类型的信息(合计，均值等)，但不允许查询单个信息

统计数据库中特殊的安全性问题：隐蔽的信息通道；能从合法的查询中推导出不合法的信息

三条规则：热河查询至少要设计N(N足够大)个以上的记录；任意两个查询的相交数据项不能超过M个任一用户的查询次数不能超过1+(N+2)/M

数据库安全机制的设计目标：使视图破坏安全的人所花费的代价远大于其所得的利益，使其放弃破坏安全

五、数据库完整性

(零)概述

数据库完整性是指数据的安全性和相容性

数据的完整性：防止数据库中存在不符合语句的数据，也就是防止数据库中存在不正确的数据；防范对象为不合语义、不正确的数据

数据的安全性：保护数据库方式恶意的破坏和非法的存取；防范的对象为非法用户和非法操作

为维护数据库的安全性，DBMS必须：提供定义完整性约束条件的机制；提供完整性检查的方法；违约处理

(一)实体完整性

1.实体完整性定义

关系模型的实体完整性：CREATE TABLE 中用PRIMARY KEY定义

单属性构成的码有两种说明方法：定义为列级约束条件；定义为表级约束条件

对多个属性构成的码只能定义为表级约束条件

主属性不能为空

e.g.将Student表中的Sno属性定义为码

在列级定义主码 在表级定义主码

CREATE TABLE Student CREATE TABLE Student

(Sno CHAR(9) PRIMARY KEY， (Sno CHAR(9)，

Sname CHAR(20) NOT NULL， Sname CHAR(20) NOT NULL，

Ssex CHAR(2) ， Ssex CHAR(2) ，

Sage SMALLINT， Sage SMALLINT，

Sdept CHAR(20)); Sdept CHAR(20)，

PRIMARY KEY (Sno) );

e.g.将SC表中的Sno，Cno属性组定义为码

CREATE TABLE SC

(Sno CHAR(9) NOT NULL，

Cno CHAR(4) NOT NULL，

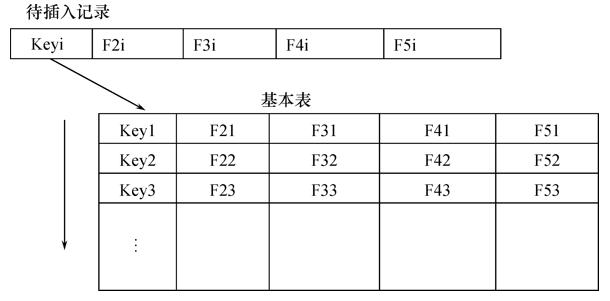
Grade SMALLINT，

PRIMARY KEY (Sno，Cno) ); /\*只能在表级定义主码\*/

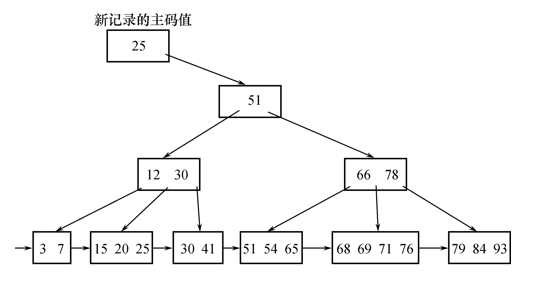
2.实体完整性检查和违约处理

请插入或对主码列进行更新操作时，RDBMS按照实体完整性规则自动进行检查。包括：检查主码值是否唯一，如果不唯一则拒绝插入或修改；检查主码的各个属性是否为空，只要有一个为空就拒绝插入或修改

检查记录中主码值是否唯一的一种方法是全表扫描：依次判断表中的每一条记录的主码值与将要插入的记录的主码值(或者修改的新主码值)是否相同



全表扫描耗时较大，关系数据库系统一般都在主码上自动建立一个索引(下图为一个B+树索引)



(二)参照完整性

SC(Sno,Cno) S(Sno)这两个关系中SC是参照关系，S是被参照关系，Sno是外码，是S的主码，(Sno，Cno)是SC的主码

1.参照完整性定义

在CREATE TABLE中用FOREIGN短语定义哪些列为外码；用REFERENCES短语知名这些外码参照哪些表的主码

e.g.关系SC中一个元组表示一个学生选修的某门课程的成绩，(Sno，Cno)是主码。Sno，Cno分别参照引用Student表的主码和Course表的主码，定义SC中的参照完整性

CREATE TABLE SC

(Sno CHAR(9) NOT NULL，

Cno CHAR(4) NOT NULL，

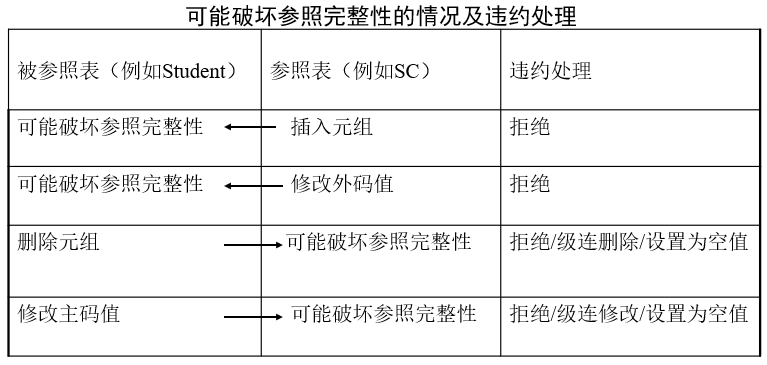
Grade SMALLINT，

PRIMARY KEY (Sno， Cno)， /\*在表级定义实体完整性\*/

FOREIGN KEY (Sno) REFERENCES Student(Sno)， /\*在表级定义参照完整性\*/

FOREIGN KEY (Cno) REFERENCES Course(Cno)); /\*在表级定义参照完整性\*/

2.参照完整性检查和违约处理



三种违约处理：拒绝执行(NO ACTION)-->默认策略；级联操作(CASCADE)-->当删除或修改被参照表的一个元组导致与参照表不一致时，删除或修改参照表中的所有导致不一致的元组；设置为空值(SET-NULL)-->对于参照完整性，除了应该定义外码，还应定义外码列是否允许为空值

e.g.显式说明参照完整性的违约处理示例

CREATE TABLE SC

(Sno CHAR(9) NOT NULL，

Cno CHAR(4) NOT NULL，

Grade SMALLINT，

PRIMARY KEY(Sno，Cno)，

FOREIGN KEY (Sno) REFERENCES Student(Sno)

ON DELETE CASCADE /\*级联删除SC表中相应的元组\*/

ON UPDATE CASCADE， /\*级联更新SC表中相应的元组\*/

FOREIGN KEY (Cno) REFERENCES Course(Cno)

ON DELETE NO ACTION /\*当删除course 表中的元组造成了与SC表不一致时拒绝删除\*/

ON UPDATE CASCADE )； /\*当更新course表中的cno时，级联更新SC表中相应的元组\*/

(三)用户定义的完整性

用户定义的完整性就是针对某一具体应用的数据必须满足的语义要求，由RDBMS提供，不必由应用程序承担

1. 属性上的约束条件的定义

CREATE TABLE时定义：列值非空(NOT NULL)；列值唯一(UNIQUE)；检查列值是否满足一个布尔表达式(CHECK)

(1)NOT NULL

e.g.在定义SC表时，说明Sno、Cno、Grade属性不允许取空值。

CREATE TABLE SC

(Sno CHAR(9) NOT NULL，

Cno CHAR(4) NOT NULL，

Grade SMALLINT NOT NULL，

PRIMARY KEY (Sno， Cno),)； /\* 如果在表级定义实体完整性，隐含了Sno，Cno不允许取空值，则在列级不允许取空值的定义就不必写了 \*/

(2)UNIQUE

e.g.建立部门表DEPT，要求部门名称Dname列取值唯一，部门编号Deptno列为主码

CREATE TABLE DEPT

(Deptno NUMERIC(2)，

Dname CHAR(9) UNIQUE，/\*要求Dname列值唯一\*/

Location CHAR(10)，

PRIMARY KEY (Deptno))；

(3)CHECK

e.g.Student表的Ssex只允许取“男”或“女”。

CREATE TABLE Student

(Sno CHAR(9) PRIMARY KEY，

Sname CHAR(8) NOT NULL，

Ssex CHAR(2) CHECK (Ssex IN (‘男’，‘女’) ) ， /\*性别属性Ssex只允许取'男'或'女' \*/

Sage SMALLINT，

Sdept CHAR(20));

2.属性上的约束条件检查和违约处理

插入元组或修改属性的值时，RDBMS检查属性上的约束条件是否被满足，如果不满足则操作被拒绝执行

3.元组上的约束条件的定义

在CREATE TABLE时可以用CHECK短语定义元组上的约束条件，即元组级约束；同属性限制想比，元组级限制可以设置不同属性之间的取值和相互约束条件

e.g.当学生的性别是男时，其名字不能以Ms.打头。

CREATE TABLE Student

(Sno CHAR(9)，

Sname CHAR(8) NOT NULL，

Ssex CHAR(2)，

Sage SMALLINT，

Sdept CHAR(20)，

PRIMARY KEY (Sno)，

CHECK (Ssex='女' OR Sname NOT LIKE 'Ms.%'))； /\*定义了元组中Sname和 Ssex两个属性值之间的约束条件\*/

性别是女性的元组都能通过该项检查，因为Ssex=‘女’成立；当性别是男性时，要通过检查则名字一定不能以Ms.打头

4.元组上的约束条件检查和违约处理

插入元组或修改属性的值时，RDBMS检查元组上的约束条件是否被满足，如果不满足则操作被拒绝执行

(四)完整性约束命名子句

1.CONSTRAINT约束

CONSTRAINT [PRIMARY KEY 短语|FOREIGN KEY|CHECK]

e.g.建立学生登记表Student，要求学号在90000~99999之间，姓名不能取空值，年龄小于30，性别只能是“男”或“女”

CREATE TABLE Student

(Sno NUMERIC(6)

CONSTRAINT C1 CHECK (Sno BETWEEN 90000 AND 99999)，

Sname CHAR(20)

CONSTRAINT C2 NOT NULL，

Sage NUMERIC(3)

CONSTRAINT C3 CHECK (Sage < 30)，

Ssex CHAR(2)

CONSTRAINT C4 CHECK (Ssex IN ( '男'，'女'))，

CONSTRAINT StudentKey PRIMARY KEY(Sno) )；

在Student表上建立了5个约束条件，包括主码约束(命名为StudentKey)以及C1、C2、C3、C4四个列级约束。

2.修改表中的完整性限制

使用ALTER TABLE语句修改表中的完整性限制

e.g.修改表Student中的约束条件，要求学号改为在900000~999999之间，年龄由小于30改为小于40

可以先删除原来的约束条件，再增加新的约束条件

ALTER TABLE Student

DROP CONSTRAINT C1;

ALTER TABLE Student

ADD CONSTRAINT C1 CHECK (Sno BETWEEN 900000 AND 999999)，

ALTER TABLE Student

DROP CONSTRAINT C3;

ALTER TABLE Student

ADD CONSTRAINT C3 CHECK (Sage < 40)；

(五)域中的完整性限制

SQL支持域的概念，并可以用CREATE DOMAIN语句建立一个域以及该域应该满足的完整性约束条件

e.g.建立一个性别域，并声明性别域的取值范围

CREATE DOMAIN GenderDomain CHAR(2)

CHECK (VALUE IN ('男'，'女') );

这样1.例子中对Ssex的说明可以改写为：Ssex GenderDomain

e.g.建立一个性别域GenderDomain，并对其中的限制命名

CREATE DOMAIN GenderDomain CHAR(2)

CONSTRAINT GD CHECK ( VALUE IN ('男'，'女') );

e.g.删除域GenderDomain的限制条件GD。

ALTER DOMAIN GenderDomain

DROP CONSTRAINT GD;

e.g.在域GenderDomain上增加限制条件GDD。

ALTER DOMAIN GenderDomain

ADD CONSTRAINT GDD CHECK (VALUE IN ( '1'，'0') );

通过后两个例子，就把性别的取值范围由('男'，'女')改为 ( '1'，'0')

(六)触发器 [第五章 数据库完整性(5)——触发器与本章小结 - 墨天轮 (modb.pro)](https://www.modb.pro/db/170667#:~:text=%E8%A7%A6%E5%8F%91%E5%99%A8%E8%A2%AB%E6%BF%80%E6%B4%BB%E6%97%B6%EF%BC%8C%E5%8F%AA%E6%9C%89%E5%BD%93%E8%A7%A6%E5%8F%91%E6%9D%A1%E4%BB%B6%E4%B8%BA%E7%9C%9F%E6%97%B6%E8%A7%A6%E5%8F%91%E5%8A%A8%E4%BD%9C%E4%BD%93%E6%89%8D%E6%89%A7%E8%A1%8C%EF%BC%8C%E5%90%A6%E5%88%99%E8%A7%A6%E5%8F%91%E5%8A%A8%E4%BD%9C%E4%BD%93%E4%B8%8D%E6%89%A7%E8%A1%8C%E3%80%82,%E5%A6%82%E6%9E%9C%E7%9C%81%E7%95%A5WHEN%E8%A7%A6%E5%8F%91%E6%9D%A1%E4%BB%B6%EF%BC%8C%E5%88%99%E8%A7%A6%E5%8F%91%E5%8A%A8%E4%BD%9C%E4%BD%93%E5%9C%A8%E8%A7%A6%E5%8F%91%E5%99%A8%E6%BF%80%E6%B4%BB%E5%90%8E%E7%AB%8B%E5%8D%B3%E6%89%A7%E8%A1%8C%E3%80%82%20%287%29%E8%A7%A6%E5%8F%91%E5%8A%A8%E4%BD%9C%E4%BD%93)

0.断言：可以定义多个表或者聚集操作的比较复杂的约束

创建断言的语句：CREATE ASSERTION 断言 CHECK 子句

CREATE ASSERTION ASS\_SC\_CNUM1

CHECK (60>=(select count(\*) FROM SC

Group by cno));

删除断言：DROP ASSERTION<断言名>

触发器是用户定义在关系表上的一类由事件驱动的特殊过程：由服务器自动激活；可以进行更为复杂的检查和操作，具有更精细和更强大的数据控制能力

1.定义触发器

语法格式：CREATE TRIGGER

{BEFORE | AFTER} ON

FOR EACH {ROW | STATEMENT}

［WHEN ］

定义触发器的语法说明:

(1)创建者：表的拥有者；

(2)触发器名；(触发器名称唯一)

(3)表名：触发器的目标表；

(4)触发事件：INSERT、DELETE、UPDATE；

(5)触发器类型： 行级触发器(FOR EACH ROW)和 语句级触发器(FOR EACH STATEMENT)

e.g.假设在TEACHER表上创建了一个AFTER UPDATE触发器。如果表TEACHER有1000行，执行如下语句：UPDATE TEACHER SET Deptno=5; 如果该触发器为语句级触发器，那么执行完该语句后，触发动作只发生一次，如果是行级触发器，触发动作将执行1000次

(6)触发条件

触发条件为真；省略WHEN触发条件

(7)触发动作体

可以是一个匿名的PL/SQl过程块，也可以是对已经创建的存储过程的调用，如果是行级触发器，用户可以在过程体中使用NEW和OLD引用UPDATE/INSERT时间之后的新值和UPDATE/INSERT事件之前的旧值；如果是语句级触发器，则不能再触发动作体中使用NEW和OLD

e.g.当对表SC的GRADE属性进行修改时，如果分数增加了10%，将此次操作记录到另一个表SC\_U(Sno,Cno,Oldgrade,Newgrade)中，Oldgrade是修改之前的分数，Newgrade是修改后的分数

CREATE TRIGGER SC\_T

AFTER UPDATE OF GRADE ON SC

REFERENCING

OLDROW AS OldTuple,

NEWROW AS Netuple

FOR EACH ROW

WHEN(NewTuple.Grade>1.1\*OldTuple.Grade)

INSERT INTO SC\_U(Sno,Cno,Oldgrade,Newgrade)

VALUES(OldTuple.Sno,OldTuple.Cno,OldTuple.Grade,NewTuple.Grade)

e.g.定义一个BEFORE行级触发器，为教师表Teacher定义完整性规则“教授的工资不得低于4000元，如果低于4000元，自动改为4000元”。

CREATE TRIGGER Insert\_Or\_Update\_Sal

BEFORE INSERT OR UPDATE ON Teacher /\*触发事件是插入或更新操作\*/

FOR EACH ROW /\*行级触发器\*/

AS BEGIN /\*定义触发动作体，是PL/SQL过程块\*/

IF (new.Job='教授') AND (new.Sal < 4000) THEN

new.Sal :=4000;

END IF;

END;

e.g. 定义AFTER行级触发器，当教师表Teacher的工资发生变化后就自动在工资变化表Sal\_log中增加一条相应记录

首先建立工资变化表Sal\_log

CREATE TABLE Sal\_log

(Eno NUMERIC(4) references teacher(eno)，

Sal NUMERIC(7，2)，

Username char(10)，

Date TIMESTAMP

)；

之后写触发器

CREATE TRIGGER Insert\_Sal

AFTER INSERT ON Teacher /\*触发事件是INSERT\*/

FOR EACH ROW

AS BEGIN

INSERT INTO Sal\_log

VALUES(new.Eno，new.Sal，CURRENT\_USER，CURRENT\_TIMESTAMP);

END;

CREATE TRIGGER Update\_Sal

AFTER UPDATE ON Teacher /\*触发事件是UPDATE \*/

FOR EACH ROW

AS BEGIN

IF (new.Sal <> old.Sal) THEN

INSERT INTO Sal\_log

VALUES(new.Eno，new.Sal，CURRENT\_USER，CURRENT\_TIMESTAMP);

END IF;

END;

2.激活触发器

触发器的执行，是由触发事件激活的，并由数据库服务器自动执行

一个数据表上可能定义了多个触发器：同一个表上的多个触发器激活时遵循如下的执行顺序：

先执行该表上的BEFORE触发器-->激活触发器的SQL语句-->执行该表上的AFTER触发器

e.g.执行修改某个教师工资的SQL语句，激活上述定义的触发器。

UPDATE Teacher SET Sal=800 WHERE Ename='陈平';

执行顺序是：

执行触发器Insert\_Or\_Update\_Sal

执行SQL语句“UPDATE Teacher SET Sal=800 WHERE Ename='陈平';”

执行触发器Insert\_Sal；

执行触发器Update\_Sal

3.删除触发器

删除触发器的SQL语法：

DROP TRIGGER ON ;

触发器必须是一个已经创建的触发器，并且只能由具有相应权限的用户删除。

e.g.删除教师表Teacher上的触发器Insert\_Sal

DROP TRIGGER Insert\_Sal ON Teacher;

六、关系数据理论

(一)问题的提出

关系数据库逻辑设计：针对具体问题，如何构造一个适合于它的数据模式；数据库逻辑设计的工具-->关系数据库的规范化理论

1.概念回顾

关系；关系模式；关系数据库；关系数据库的模式

2.关系模式的形式化定义

关系模式由五部分组成，即它是一个五元组：R(U, D, DOM, F)

R：关系名；U：组成该关系的属性名集合；D：属性组U中属性所来自的域；DOM：属性向域的映象集合；F：属性间数据的依赖关系集合

3.什么是数据依赖

(1)完整性约束的表现形式

限定属性取值范围：例如学生成绩必须在0-100之间；定义属性值间的相互关连(主要体现于值的相等与否)，这就是数据依赖，它是数据库模式设计的关键

(2)数据依赖

一个关系内部属性与属性之间的约束关系；现实世界属性间相互联系的抽象；数据内在的性质；语义的体现

(3)数据依赖的类型

函数依赖(Functional Dependency，简记为FD);多值依赖(Multivalued Dependency，简记为MVD);其他

4.关系模式的简化定义

关系模式R(U, D, DOM, F)可以简化为一个三元组：R(U, F)。当且仅当U上的一个关系r满足F时，r称为关系模式 R(U, F)的一个关系

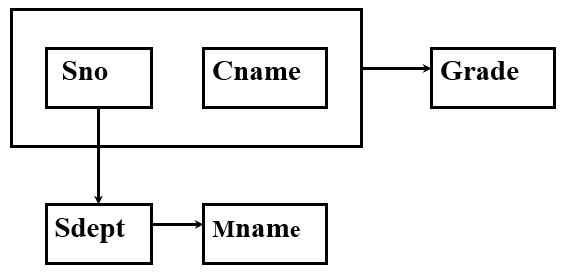
5.数据依赖对关系模式的影响

e.g.建立一个描述学校教务的数据库：

学生的学号(Sno)、所在系(Sdept)、系主任姓名(Mname)、课程名(Cname)、成绩(Grade)

单一的关系模式：Student U＝{ Sno, Sdept, Mname, Cname, Grade ｝

属性组U上的一组函数依赖F：F ＝｛ Sno → Sdept, Sdept → Mname, (Sno, Cname) → Grade ｝



关系模式Student中存在的问题： 数据冗余太大；更新异常(Update Anomalies)；插入异常(Insertion Anomalies)；删除异常(Deletion Anomalies)

结论：Student关系模式不是一个好的模式。

“好”的模式：不会发生插入异常、删除异常、更新异常，数据冗余应尽可能少

原因：由存在于模式中的某些数据依赖引起的

解决方法：通过分解关系模式来消除其中不合适的数据依赖

把这个单一模式分成3个关系模式：

S(Sno，Sdept，Sno → Sdept);SC(Sno，Cno，Grade，(Sno，Cno) → Grade);DEPT(Sdept，name，Sdept→ Mname)

(二)规范化

规范化理论是用来改造关系模式，通过分解关系模式来消除其中不合适的数据依赖，以解决插入异常、产出异常、更新异常和数据冗余的问题

1.函数依赖

(1)函数依赖

定义6.1 设R(U)是一个属性集U上的关系模式，X和Y是U的子集。

若对于R(U)的任意一个可能的关系r，r中不可能存在两个元组在X上的属性值相等， 而在Y上的属性值不等， 则称 “X函数确定Y” 或 “Y函数依赖于X”，记作X→Y。

说明：所有关系实例均要满足；语义范畴的概念；数据库设计者可以对现实世界作强制的规定

(2)平凡函数依赖与非平凡函数依赖

在关系模式R(U)中，对于U的子集X和Y，

如果X→Y，但Y 不⊆ X，则称X→Y是非平凡的函数依赖

如果X→Y，但Y ⊆ X, 则称X→Y是平凡的函数依赖

例：在关系SC(Sno, Cno, Grade)中，

非平凡函数依赖： (Sno, Cno) → Grade

平凡函数依赖： (Sno, Cno) → Sno 和 (Sno, Cno) → Cno

若X→Y，则X称为这个函数依赖的决定属性组，也称为决定因素(Determinant)。

若X→Y，Y→X，则记作X←→Y。

若Y不函数依赖于X，则记作X\→Y。

(3)完全函数依赖与部分函数依赖

定义6.2 在R(U)中，如果X→Y，并且对于X的任何一个真子集X’，都有X’ \→ Y, 则称Y对X完全函数依赖，记作X →F Y。 若X→Y，但Y不完全函数依赖于X，则称Y对X部分函数依赖，记作X → P Y

(4)传递函数依赖

定义6.3 在R(U)中，如果X→Y，(Y 不包含 X) ,Y→X Y→Z， 则称Z对X传递函数依赖。 记为：X → Z

注: 如果Y→X， 即X←→Y，则Z直接依赖于X。

例: 在关系Std(Sno, Sdept, Mname)中，有：Sno → Sdept，Sdept → Mname。Mname传递函数依赖于Sno

2.码

定义6.4 设K为R中的属性或属性组合。若K🡪(F)U，则K称为R的侯选码(Candidate Key)。 注：U是完全依赖于K，而不是部分函数依赖于K，一般的如果U依赖于K，即K🡪U，则称K为超码

若候选码多于一个，则选定其中的一个做为主码(Primary Key)。

主属性与非主属性：包含在任何一个候选码中的属性 ，称为主属性(Prime attribute) ；不包含在任何码中的属性称为非主属性(Nonprime attribute)或非码属性(Non-key attribute)

全码：整个属性组是码，称为全码(All-key)

e.g.关系模式S(Sno,Sdept,Sage)，单个属性Sno是码，SC(Sno，Cno，Grade)中，(Sno，Cno)是码

关系模式R(P，W，A)，P：演奏者 W：作品 A：听众 一个演奏者可以演奏多个作品，某一作品可被多个演奏者演奏，听众可以欣赏不同演奏者的不同作品，所以码为(P，W，A)，即All-Key

外码：定义6.5 关系模式 R 中属性或属性组X 并非 R的码，但 X 是另一个关系模式的码，则称 X 是R 的外部码(Foreign key)也称外码

如在SC(Sno，Cno，Grade)中，Sno不是码，但Sno是关系模式S(Sno，Sdept，Sage)的码，则Sno是关系模式SC的外部码

主码与外部码一起提供了表示关系间联系的手段

3.范式

范式是符合某一种级别的关系模式的集合

关系数据库中的关系必须满足一定的要求。满足不同程度要求的为不同范式

范式的种类： 第一范式(1NF)；第二范式(2NF)；第三范式(3NF)；BC范式(BCNF)；第四范式(4NF)；第五范式(5NF)

各种范式之间存在联系：

0

某一关系模式R为第n范式，可简记为R∈nNF。

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，这种过程就叫规范化

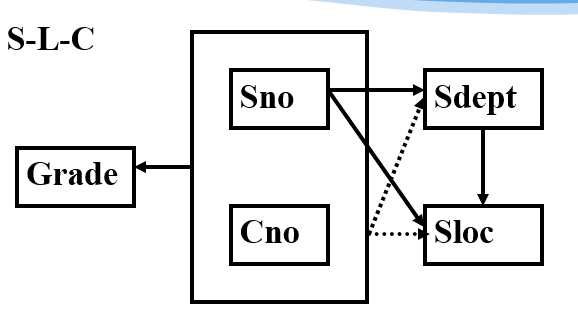
4.2NF(第二范式)

(1)1NF的定义

如果一个关系模式R的所有属性都是不可分割的基本数据项，则R∈1NF

第一范式是对关系模式的最起码要求，不满足第一范式的数据库模式不能成为关系模式；但是满足第一范式的关系模式不一定是一个好的关系模式

e.g.关系模式 S-L-C(Sno, Sdept, Sloc, Cno, Grade) Sloc为学生住处，假设每个系的学生住在同一个地方



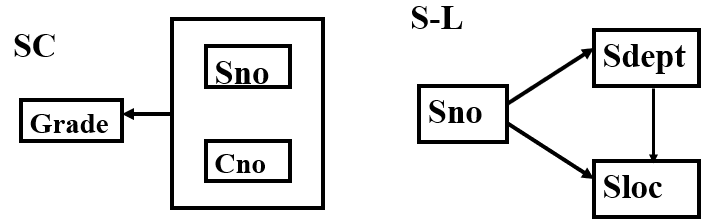
函数依赖包括：(Sno, Cno) → F Grade；Sno → Sdept；(Sno, Cno) →P Sdept；Sno → Sloc；(Sno, Cno) →P Sloc；Sdept → Sloc

S-L-C的码为(Sno, Cno)；S-L-C满足第一范式；非主属性Sdept和Sloc部分函数依赖于码(Sno, Cno)

S\_L\_C不是一个好模式：插入异常；删除异常；修改复杂；数据冗余度大

原因：Sdept、Sloc部分函数依赖于码

解决方法：S\_L\_C分解为两个关系模式，以消除这些部分函数依赖：SC(Sno， Cno， Grade)S-L(Sno， Sdept， Sloc)



关系模式SC的码为(Sno,Cno)；关系模式S\_L的码为Sno；这样非主属性对码都是完全函数依赖

(2)2NF的定义(非主属性没有部分函数依赖)

定义6.6 若R∈1NF，且每一个非主属性完全函数依赖于码，则R∈2NF。

例：S-L-C(Sno, Sdept, Sloc, Cno, Grade) ∈1NF

S-L-C(Sno, Sdept, Sloc, Cno, Grade) \∈2NF

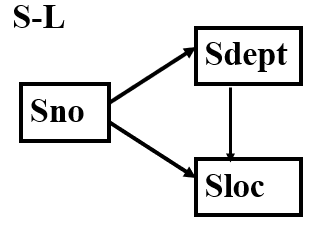
SC(Sno， Cno， Grade) ∈ 2NF

S-L(Sno， Sdept， Sloc) ∈ 2NF

采用投影分解法将一个1NF的关系分解为多个2NF的关系，可以在一定程度上减轻原1NF关系中存在的插入异常、删除异常、数据冗余度大、修改复杂等问题；将一个1NF关系分解为多个2NF的关系，并不能完全消除关系模式中的各种异常情况和数据冗余

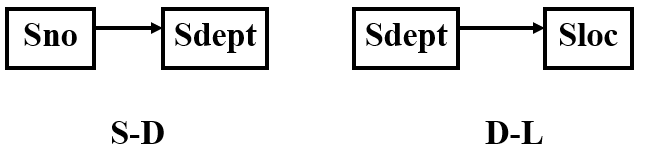
5.3NF(第三范式)(没有非主属性最低3NF)(非主属性没有部分依赖，没有传递依赖)

定义6.7 关系模式R 中若不存在这样的码X、属性组Y及非主属性Z(Z 不⊆ Y), 使得X→Y，Y→Z成立， Y → X，则称R ∈ 3NF。若R∈3NF，则每一个非主属性既不部分依赖于码也不传递依赖于码。

e.g.2NF关系模式S-L(Sno, Sdept, Sloc)中 函数依赖： Sno→Sdept Sdept \→ Sno Sdept→Sloc 可得：Sno→传递Sloc，即S-L中存在非主属性对码的传递函数依赖，S-L \∈ 3NF

解决方法: 采用投影分解法，把S-L分解为两个关系模式，以消除传递函数依赖：S-D(Sno， Sdept)；D-L(Sdept，Sloc)；S-D的码为Sno， D-L的码为Sdept。分解后的关系模式S-D与D-L中不再存在传递依赖

S-L(Sno, Sdept, Sloc) ∈ 2NF S-L(Sno, Sdept, Sloc) \∈ 3NF S-D(Sno，Sdept) ∈ 3NF D-L(Sdept， Sloc)∈ 3NF



采用投影分解法将一个2NF的关系分解为多个3NF的关系，可以在一定程度上解决原2NF关系中存在的插入异常、删除异常、数据冗余度大、修改复杂等问题。将一个2NF关系分解为多个3NF的关系后，仍然不能完全消除关系模式中的各种异常情况和数据冗余。

6.BCNF(BC范式)(所有决定因素都是码)

定义6.8 关系模式R∈1NF，若X→Y且Y 不⊆ X时X必含有码，则R ∈BCNF。

等价于：每一个决定属性因素都包含码(每一个决定因素都是码)

若R∈BCNF则所有非主属性对每一个码都是完全函数依赖；所有的主属性对每一个不包含它的码，也是完全函数依赖；没有任何属性完全函数依赖于非码的任何一组属性

关系模式C(Cno,Cname,Pcno)：C∈3NF C∈BCNF

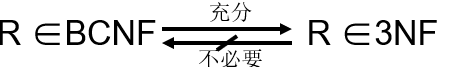
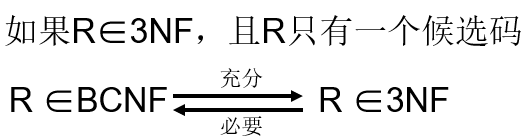
关系模式S(Sno,Sname,Sdept,Sage)：假定S有两个码Sno,Sname S∈3NF S∈BCNF

关系模式SJP(S，J，P) 函数依赖：(S，J)→P；(J，P)→S (S，J)与(J，P)都可以作为候选码,属性相交。SJP∈3NF，SJP∈BCNF

在关系模式STJ(S，T，J)中，S表示学生，T表示教师，J表示课程。函数依赖：(S，J)→T，(S，T)→J，T→J (S，J)和(S，T)都是候选码

STJ∈3NF：没有任何非主属性对码传递依赖或部分依赖 ；STJ∈BCNF：T是决定因素，T不包含码

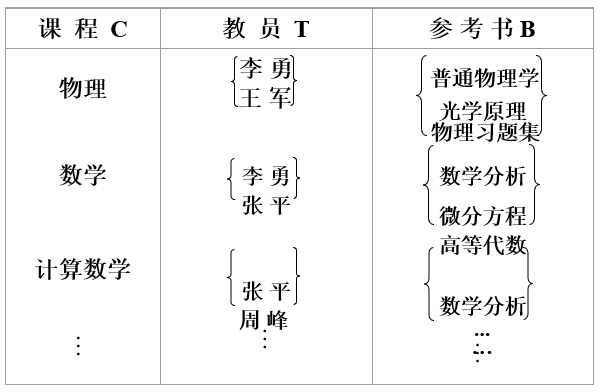
解决方法：将STJ分解为二个关系模式：ST(S，T) ∈ BCNF， TJ(T，J)∈ BCNF 没有任何属性对码的部分函数依赖和传递函数依赖

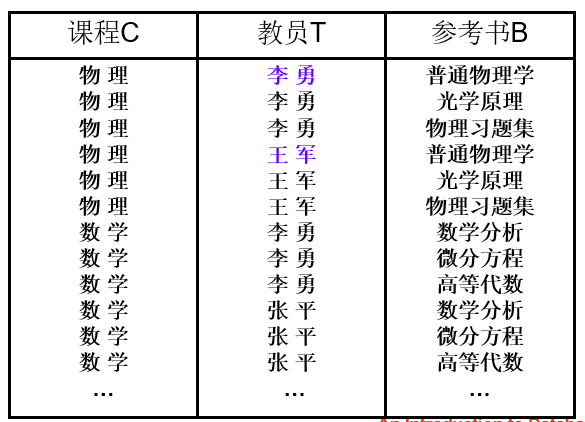
7. 多值依赖(了解)

(0)引入

学校中某一门课程由多个教师讲授，他们使用相同的一套参考书。每个教员可以讲授多门课程，每种参考书可以供多门课程使用。

非规范化关系：

用二维表表示Teaching



Teaching∈BCNF；Teaching具有唯一候选码(C，T，B)， 即全码；Teaching模式中存在的问题：数据冗余度大；插入操作复杂；删除操作复杂；修改操作复杂

1. 定义

设R(U)是一个属性集U上的一个关系模式， X、 Y和Z是U的子集，并且Z＝U－X－Y。关系模式R(U)中多值依赖 X→→Y成立，当且仅当对R(U)的任一关系r，给定的一对(x，z)值，有一组Y的值，这组值仅仅决定于x值而与z值无关 例：Teaching(C, T, B)

多值依赖的另一个等价的形式化的定义： 在R(U)的任一关系r中，如果存在元组t，s 使得t[X]=s[X]，那么就必然存在元组 w，v∈ r，(w，v可以与s，t相同)，使得w[X]=v[X]=t[X]，而w[Y]=t[Y]，w[Z]=s[Z]，v[Y]=s[Y]，v[Z]=t[Z](即交换s，t元组的Y值所得的两个新元组必在r中)，则Y多值依赖于X，记为X→→Y。 这里，X，Y是U的子集，Z=U-X-Y。

平凡多值依赖和非平凡的多值依赖：若X→→Y，而Z＝φ，则称X→→Y为平凡的多值依赖，否则称X→→Y为非平凡的多值依赖

(2)多值依赖的性质

对称性：若X→→Y，则X→→Z，其中Z＝U－X－Y

传递性：若X→→Y，Y→→Z， 则X→→Z –Y

函数依赖是多值依赖的特殊情况：若X→Y，则X→→Y

若X→→Y，X→→Z，则X→→Y∪ Z

若X→→Y，X→→Z，则X→→Y∩Z

若X→→Y，X→→Z，则X→→Y-Z，X→→Z -Y

多值依赖的有效性与属性集的范围有关

若函数依赖X→Y在R(U)上成立，则对于任何Y' ⊆ Y均有X→Y' 成立

多值依赖X→→Y若在R(U)上成立，不能断言对于任何Y⊆ Y有X→→Y' 成立

8.4NF (第四范式)(了解)

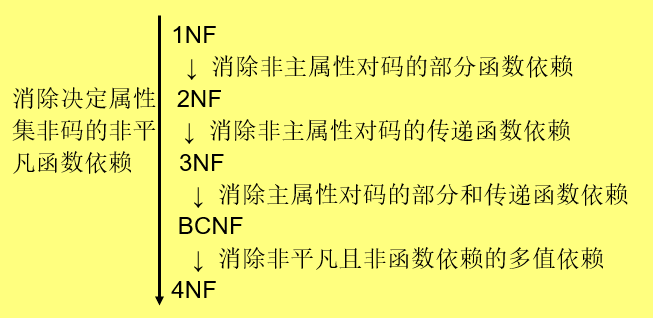
定义6.10 关系模式R<U，F>∈1NF，如果对于R的每个非平凡多值依赖X→→Y(Y 不⊆ X)，X都含有码，则R∈4NF。

如果R ∈ 4NF， 则R ∈ BCNF不允许有非平凡且非函数依赖的多值依赖；允许的非平凡多值依赖是函数依赖

e.g. Teaching(C,T,B) ∈ 4NF存在非平凡的多值依赖C→→T，且C不是码

用投影分解法把Teaching分解为如下两个关系模式：CT(C, T) ∈ 4NF；CB(C, B) ∈ 4NF。C→→T， C→→B是平凡多值依赖

9.规范化小结

 关系数据库的规范化理论是数据库逻辑设计的工具；目的：尽量消除插入、删除一场，修改复杂，数据冗余；基本思想：逐步消除数据依赖中不合适的部分；实质：概念的单一化

关系模式规范化的基本步骤：

不能说规范化程度越高的关系模式就越好；在设计数据库模式结构时，必须对现实世界的实际情况和用户应用需求作进一步分析，确定一个合适的、能够反映现实世界的模式；上面的规范化步骤可以在其中任何一步终止

(三)数据依赖的公理系统

1.Armstrong公理系统

逻辑蕴含：定义6.11 对于满足一组函数依赖F的关系模式R <U，F>，其任何一个关系r，若函数依赖X→Y都成立, (即r中任意两元组t，s，若t[X］=s[X］，则t[Y］=s[Y］)，则称F逻辑蕴含X →Y

关系模式R<U,F>来说有以下推理规则：

A1. 自反律(Reflexivity)：若Y ⊆ X ⊆ U，则X →Y为F所蕴含。

A2.增广律(Augmentation)：若X→Y为F所蕴含，且Z ⊆ U，则XZ→YZ为F所蕴含。

A3.传递律(Transitivity)：若X→Y及Y→Z为F所蕴含，则X→Z为F所蕴含。

2.导出规则

根据A1，A2，A3这三条推理规则可以得到下面三条推理规则：

合并规则：由X→Y，X→Z，有X→YZ。 (A2， A3)

伪传递规则：由X→Y，WY→Z，有XW→Z。 (A2， A3)

分解规则：由X→Y及 Z⊆Y，有X→Z。 (A1， A3)

根据合并规则和分解规则，可得引理6.1：X→A1 A2…Ak成立的充分必要条件是X→Ai成立(i=l，2，…，k)

Armstrong公理系统是有效的、完备的

有效性：由F出发根据Armstrong公理推导出来的每一个函数依赖一定在F+中；

完备性：F+中的每一个函数依赖，必定可以由F出发根据Armstrong公理推导出来

3.函数依赖闭包

定义6.l2 在关系模式R<U，F>中为F所逻辑蕴含的函数依赖的全体叫作 F的闭包，记为F+。

定义6.13 设F为属性集U上的一组函数依赖，X U， XF+ ={ A|X→A能由F 根据Armstrong公理导出}，XF+称为属性集X关于函数依赖集F的闭包

引理6.2 设F为属性集U上的一组函数依赖，X，Y ⊆ U，X→Y能由F 根据Armstrong公理导出的充分必要条件是Y ⊆ XF+

用途：将判定X→Y是否能由F根据Armstrong公理导出的问题，转化为求出XF+ 、判定Y是否为XF+的子集的问题

算法6.1 求属性集X(X ⊆ U)关于U上的函数依赖集F 的闭包XF+

输入：X，F 输出：XF+

步骤：

(1)令X(0)=X，i=0

(2)求B，这里

(3)X(i+1)=B∪X(i)

(4)判断X(i+1)= X (i)吗?

(5)若相等或X(i)=U , 则X(i)就是XF+ , 算法终止。

(6)若否，则 i=i+l，返回第(2)步。

对于算法6.1， 令ai =|X(i)|，{ai }形成一个步长大于1的严格递增的序列，序列的上界是 | U |，因此该算法最多 |U| - |X| 次循环就会终止。

e.g.已知关系模式R<U，F>，其中U={A，B，C，D，E}；F={AB→C，B→D，C→E，EC→B，AC→B}。求(AB)F+ 。

解 设X(0)=AB；

(1) X(1)=AB∪CD=ABCD。

(2) X(0)≠ X(1)

X(2)=X(1)∪BE=ABCDE。

(3) X(2)=U，算法终止

→(AB)F+ =ABCDE。

4.Armstrong公里系统放入有效性和完备性

定理6.2 Armstrong公理系统是有效的、完备的

证明：

有效性：可由定理6.1得证

完备性：只需证明逆否命题: 若函数依赖X→Y不能由F从Armstrong公理导出，那么它必然不为F所蕴含

(1) 引理: 若V→W成立，且V ⊆ XF+，则W ⊆ XF+

(2) 构造一张二维表r，它由下列两个元组构成，可以证明r必是R(U，F)的一个关系，即F+中的全部函数依赖在 r上成立。

XF+ U-XF+

11......1 00......0

11......1 11......1

(3) 若X→Y 不能由F从Armstrong公理导出，则Y 不是XF+ 的子集。

5.函数依赖集等价

定义6.14 如果G+=F+，就说函数依赖集F覆盖G(F是G的覆盖，或G是F的覆盖)，或F与G等价。

引理6.3 F+ = G+ 的充分必要条件是F ⊆ G+ ，和G ⊆ F+

证: 必要性显然，只证充分性。

(1)若F⊆G+ ，则XF+ ⊆ XG++ 。

(2)任取X→Y∈F+ 则有 Y ⊆ XF+  ⊆ XG++ 。

所以X→Y ∈ (G+)+= G+。即F+ ⊆ G+。

(3)同理可证G+ ⊆ F+ ，所以F+ = G+。

6.最小依赖集

定义6.15 如果函数依赖集F满足下列条件，则称F为一个极小函数依赖集。亦称为最小依赖集或最小覆盖。

(1) F中任一函数依赖的右部仅含有一个属性。

(2) F中不存在这样的函数依赖X→A，使得F与F-{X→A}等价。

(3) F中不存在这样的函数依赖X→A， X有真子集Z使得F-{X→A}∪{Z→A}与F等价。

e.g.关系模式S<U，F>，其中：U={ Sno，Sdept，Mname，Cno，Grade }，F={ Sno→Sdept，Sdept→Mname，(Sno，Cno)→Grade }， 设F’={Sno→Sdept，Sno→Mname，Sdept→Mname，(Sno，Cno)→Grade，(Sno，Sdept)→Sdept}

F是最小覆盖，而F’不是。

因为：F ’ - {Sno→Mname}与F ’等价

F ’ - {(Sno，Sdept)→Sdept}也与F ’等价

7.极小化过程

定理6.3 每一个函数依赖集F均等价于一个极小函数依赖

集Fm。此Fm称为F的最小依赖集。

证明: 构造性证明，找出F的一个最小依赖集。

(1)逐一检查F中各函数依赖FDi：X→Y，若Y=A1A2 …Ak，k > 2，

则用 { X→Aj |j=1，2，…， k} 来取代X→Y。

(2)逐一检查F中各函数依赖FDi：X→A，令G=F-{X→A}，

若A∈XG+， 则从F中去掉此函数依赖。

(3)逐一取出F中各函数依赖FDi：X→A，设X=B1B2…Bm，

逐一考查Bi (i=l，2，…，m)，若A ∈(X-Bi )F+ ，

则以X-Bi 取代X。

e.g. F = {A→B，B→A，B→C，A→C，C→A}，Fm1、Fm2都是F的最小依赖集：Fm1= {A→B，B→C，C→A}  Fm2= {A→B，B→A，A→C，C→A} ；

F的最小依赖集Fm不唯一

极小化过程( 定理6.3的证明 )也是检验F是否为极小依赖集的一个算法

(四)模式的分解

1.诸多定义

把低一级的关系模式分解为若干个高一级的关系模式的方法不是唯一的；只有能够保证分解后的关系模式与原关系模式等价，分解方法才有意义

三种模式分解等价的定义：分解具有无损连接性；分解要保持函数依赖；分解既要保持函数依赖，又要具有无损连接性

定义6.16 关系模式R<U,F>的一个分解：ρ={ R1<U1,F1>，R2<U2,F2>，…，Rn<Un,Fn>} U= ∪Ui，且不存在 Ui ⊆ Uj，Fi 为 F在 Ui 上的投影

定义6.17 函数依赖集合{X→Y | X→Y ∈ F+∧XY⊆Ui} 的一个覆盖 Fi 叫作 F 在属性 Ui 上的投影

关系模式R<U,F>的一个分解 ρ={ R1<U1,F1>，R2<U2,F2>， …，Rn<Un,Fn>}

若R与R1、R2、…、Rn自然连接的结果相等，则称关系模式R的这个分解ρ具有无损连接性(Lossless join)

R1∩R2→R1-R2 或者 R1∩R2→R2-R1 (R-S ={ t|t∈R∧t\∈S })

具有无损连接性的分解保证不丢失信息；无损连接性不一定能解决插入异常、删除异常、修改复杂、数据冗余等问题

设关系模式R<U,F>被分解为若干个关系模式R1<U1,F1>，R2<U2,F2>，…，Rn<Un,Fn>

(其中U=U1∪U2∪…∪Un，且不存在Ui ⊆ Uj，Fi为F在Ui上的投影)，若F所逻辑蕴含的函数依赖一定也由分解得到的某个关系模式中的函数依赖Fi所逻辑蕴含，则称关系模式R的这个分解是保持函数依赖的(Preserve dependency)

2.一个分解例子

e.g. 例：S-L(Sno， Sdept， Sloc)F={ Sno→Sdept,Sdept→Sloc,Sno→Sloc} S-L∈2NF

分解方法可以有多种：

1.S-L分解为三个关系模式：SN(Sno) SD(Sdept) SO(Sloc)

2.SL分解为下面二个关系模式： NL(Sno, Sloc) DL(Sdept, Sloc)

3.将SL分解为下面二个关系模式： ND(Sno, Sdept) NL(Sno, Sloc)

4.将SL分解为下面二个关系模式： ND(Sno, Sdept) DL(Sdept, Sloc)

第1种分解方法既不具有无损连接性，也未保持函数依赖，

它不是原关系模式的一个等价分解

第2种分解方法保持了函数依赖，但不具有无损连接性

第3种分解方法具有无损连接性，但未持函数依赖Sdept→Sloc没有投影到关系ND、NL上

第4种分解方法既具有无损连接性，又保持了函数依赖

3.分解的种类

如果一个分解具有无损连接性，则它能够保证不丢失信息

如果一个分解保持了函数依赖，则它可以减轻或解决各种异常情况

分解具有无损连接性和分解保持函数依赖是两个互相独立的标准。具有无损连接性的分解不一定能够保持函数依赖；同样，保持函数依赖的分解也不一定具有无损连接性。

4.书上的几个算法

算法6.2 判别一个分解的无损连接性 p.197

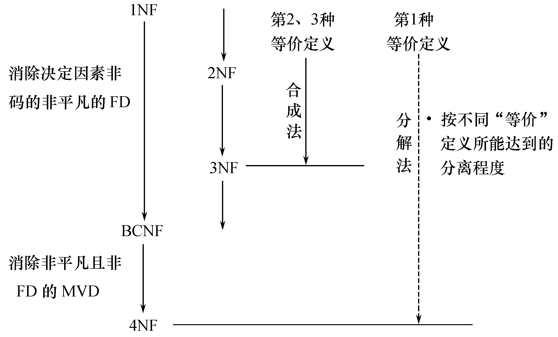
算法6.3(合成法)转换为3NF的保持函数依赖的分解。 P.198

算法6.4 转换为3NF既有无损连接性又保持函数依赖的分解 p.199

算法6.5 (分解法)转换为BCNF的无损连接分解 p.199

算法6.6 达到4NF的具有无损连接性的分解 p.201

(五)小结

关系模式的规范化，基本思想：

若要求分解具有无损连接性，那么模式分解一定能够达到4NF

若要求分解保持函数依赖，那么模式分解一定能够达到3NF，但不一定能够达到BCNF

若要求分解既具有无损连接性，又保持函数依赖，则模式分解一定能够达到3NF，但不一定能够达到BCNF

规范化理论为数据库设计提供了理论的指南和工具,也仅仅是指南和工具

并不是规范化程度越高，模式就越好,必须结合应用环境和现实世界的具体情况合理地选择数据库模式

**七、数据库设计(另一个课件)**

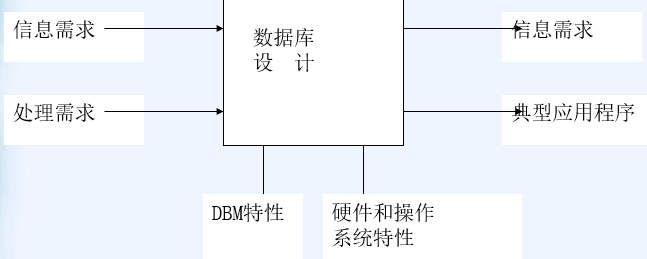
(一)数据库设计概述

1.数据库设计的任务、内容和特点

(1)数据库设计的任务

数据库设计是指根据用户需求研制数据库结构的过程，具体的说是指对于一个给定的应用环境，构造最有的数据库模式，建立数据库及其应用系统，是指能有效的存储数据，满足用户的信息要求和处理要求；也就是把现实世界的数据根据何种应用处理的要求，加一合理地组织，满足硬件和操作系统的特性，利用已有的DBMS来建立能够实现系统目标的数据库

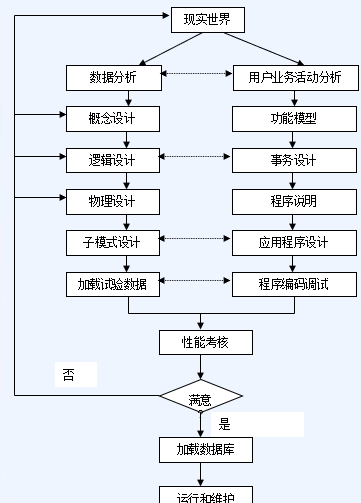
数据库设计的任务：



(2)数据库设计的内容

数据库设计包括数据库的结构设计和数据库的行为设计两方面的内容

a.数据库的结构设计

 数据库的结构设计是指根据给定的应用环境进行数据库的模式或子模式的设计，包括数据库的概念设计、逻辑设计和物理设计。数据库模式是各应用程序共享的结构，是静态的、稳定的，一经形成后通常情况下是不容易改变的，所以结构设计又称为静态模型设计

b.数据库的行为设计

数据库的行为设计是指确定数据库用户的行为和动作，而在数据库系统中，用户的行为和动作是指用户对于数据库的操作，这些要通过应用程序来实现，所以数据库的行为设计就是应用程序的设计。用户的行为总是使数据库的内容发生变化，所以行为设计是动态的，所以行为设计又称为动态模型设计

(3)数据库设计的特点

在70年代末80年代初，人们为了研究数据库设计方法学的便利，曾主张将结构设计和行为设计两者分离，随着数据库设计方法学的成熟和结构化分析、设计方法的普遍使用，人们主张将两者作一体化的考虑，这样可以缩短数据库的设计周期，提高数据库的设计效率。

现代数据库的设计的特点是强调结构设计与行为设计相结合，是一种“反复探寻，逐步求精”的过程。首先从数据模型开始设计，以数据模型为核心进行展开，数据库设计和应用系统设计相结合，建立一个完整、独立、共享、冗余小、安全有效的数据库系统。上图给出了数据库设计的全过程

2.数据库设计方法简述

数据库设计方法目前可分为四类直观设计法、规范设计法、计算机辅助设计法和自动化设计方法

直观设计法也叫手工试凑法，它是最早使用的数据库设计方法。这种方法依赖于设计者的经验和技巧，缺乏科学理论和工程原则的支持，设计的质量很难保证，常常是数据库运行一段时间后又发现各种问题，这样再重新进行修改，增加了系统维护的代价。因此这种方法越来越不适应信息管理发展的需要。

为了改变这种情况，1978年10月，来自三十多个国家的数据库专家在美国新奥尔良(New Orleans)市专门讨论了数据库设计问题，他们运用软件工程的思想和方法，提出了数据库设计的规范，这就是著名的新奥尔良法，它是目前公认的比较完整和权威的一种规范设计法。新奥尔良法将数据库设计分成需求分析(分析用户需求)、概念设计(信息分析和定义)、逻辑设计(设计实现)和物理设计(物理数据库设计)。目前，常用的规范设计方法大多起源于新奥尔良法，并在设计的每一阶段采用一些辅助方法来具体实现。

a.基于E-R模型的数据库设计方法

由P.P.S.chen于1976年提出的数据库设计方法，其基本思想是在需求分析的基础上，用E-R(实体—联系)图构造一个反映现实世界实体之间联系的企业模式，然后再将此企业模式转换成基于某一特定的DBMS的概念模式。

b. 基于3NF的数据库设计方法

由S·Atre提出的结构化设计方法，其基本思想是在需求分析的基础上，确定数据库模式中的全部属性和属性间的依赖关系，将它们组织在一个单一的关系模式中，然后再分析模式中不符合3NF的约束条件，将其进行投影分解，规范成若干个3NF关系模式的集合。具体分为5个步骤：(1)设计企业模式，利用规范化得到的3NF关系模式画出企业模式；(2)设计数据库的概念模式，把企业模式转换成DBMS所能接受的概念模式，并根据概念模式导出各个应用的外模式；(3)设计数据库的物理模式(存储模式)；

(4)对物理模式进行评价； (5) 实现数据库。

c. 此方法先从分析各个应用的数据着手，其基本思想是为每个应用建立自己的视图，然后再把这些视图汇总起来合并成整个数据库的概念模式。合并过程中要解决以下问题：(1) 消除命名冲突；(2) 消除冗余的实体和联系；(3)进行模式重构，在消除了命名冲突和冗余后，需要对整个汇总模式进行调整，使其满足全部完整性约束条件

除了以上三种方法外，规范化设计方法还有实体分析法、属性分析法和基于抽象语义的设计方法等，这里不再详细介绍。规范设计法从本质上来说仍然是手工设计方法，其基本思想是过程迭代和逐步求精。计算机辅助设计法是指在数据库设计的某些过程中模拟某一规范化设计的方法，并以人的知识或经验为主导，通过人机交互方式实现设计中的某些部分。

目前许多计算机辅助软件工程(Computer Aided Software Engineering，CASE)工具可以自动或辅助设计人员完成数据库设计过程中的很多任务.比如SYSBASE公司的PowerDesigner和Oracle公司的Design 2000。

3.数据库设计的步骤

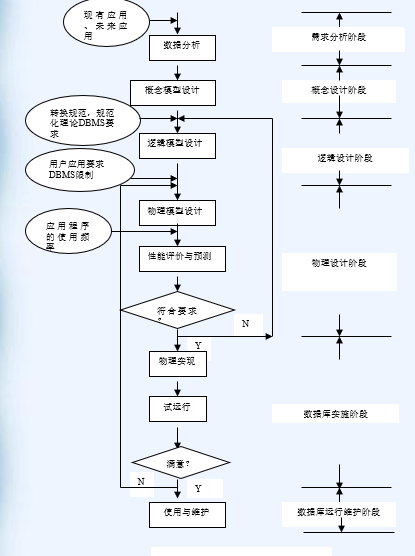
和其他软件一样，数据库的设计过程可以使用软件工程中的生存周期的概念来说明，称为“数据库设计的生存期”，它是指从数据库研制到不再使用它的整个时期。

按规范设计法可将数据库设计分为六个阶段：(1)系统需求分析阶段(2)概念结构设计阶段(3)逻辑结构设计阶段(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

该方法是分阶段完成的，每完成一个阶段，都要进行设计分析，评价一些重要的设计指标，把设计阶段产生的文档组织评审，与用户进行交流。如果设计的数据库不符合要求则进行修改，这种分析和修改可能要重复若干次，以求最后实现的数据库能够比较精确地模拟现实世界，能较准确地反映用户的需求，设计一个完善的数据库应用系统往往是六个阶段的不断反复的过程。

数据库设计中，前两个阶段是面向用户的应用要求，面向具体的问题；中间两个阶段是面向数据库管理系统；最后两个阶段是面向具体的实现方法。前四个阶段可统称为“分析和设计阶段”，后两个阶段称为“实现和运行阶段”。

六个阶段的主要工作各有不同。

a.系统需求分析阶段

需求分析是整个数据库设计过程的基础，要收集数据库所有用户的信息内容和处理要求，并加以规格化和分析。这是最费时、最复杂的一步，但也是最重要的一步，相当于待构建的数据库大厦的地基，它决定了以后各步设计的速度与质量。需求分析做得不好，可能会导致整个数据库设计返工重做。在分析用户需求时，要确保用户目标的一致性。

b.概念结构设计阶段

概念设计是把用户的信息要求统一到一个整体逻辑结构中，此结构能够表达用户的要求，是一个独立于任何DBMS软件和硬件的概念模型。

c.逻辑结构设计阶段

逻辑设计是将上一步所得到的概念模型转换为某个DBMS所支持的数据模型，并对其进行优化。

d.物理设计阶段

物理设计是为逻辑数据模型建立一个完整的能实现的数据库结构，包括存储结构和存取方法。

上述分析和设计阶段是很重要的，如果做出不恰当的分析或设计，则会导致一个不恰当或反应迟钝的应用系统。

e.数据库实施阶段

根据物理设计的结果把原始数据装入数据库，建立一个具体的数据库并编写和调试相应的应用程序。应用程序的开发目标是开发一个可依赖的有效的数据库存取程序，来满足用户的处理要求。

f.数据库运行与维护阶段

这一阶段主要是收集和记录实际系统运行的数据，数据库运行的记录用来提高用户要求的有效信息，用来评价数据库系统的性能，进一步调整和修改数据库。在运行中，必须保持数据库的完整性，并能有效地处理数据库故障和进行数据库恢复。在运行和维护阶段，可能要对数据库结构进行修改或扩充。

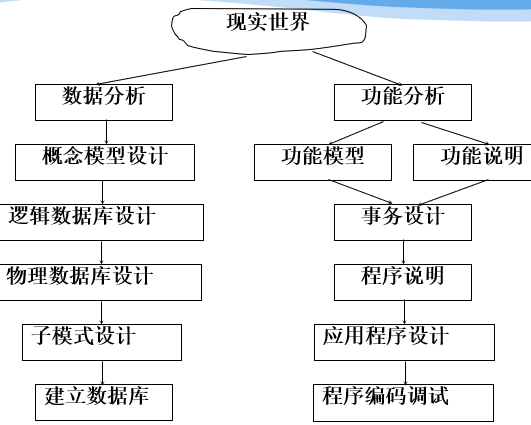
**七、数据库设计**

(一)数据库设计概述

数据库设计：数据库设计是指对于一个给定的应用环境，构造(设计)优化的数据库逻辑模式和物理结构，并据此建立数据库及其应用系统，使之能够有效的存储和管理数据，满足各种用户的应用需求，包括信息管理要求和数据操作要求。目标是为用户和各种应用系统提供搞一个信息基础设施和高效率的运行环境

1.数据库设计的特点

数据库建设的基本规律：三分技术，七分管理，十二分基础数据(管理：数据库建设项目管理；企业(应用部门)的业务管理；基础数据：收集、入库；更新新的数据)

结构(数据)设计和行为(处理)设计相结合：将数据库结构设计和数据处理设计密切结合

2.数据库设计方法

手工与经验相结合方法：设计质量与设计人员的经验和水平有直接关系；数据库迎新一段时间后常常不同程度的发现各种问题，增加了维护代价

规范设计法：基本思想-->过程迭代和逐步求精

新奥尔良方法：将数据库设计分为若干阶段和步骤

基于E-R模型的数据库设计方法：概念设计阶段广泛应用

3NF(第三范式)的设计方法：逻辑阶段可采用的有效方法

ODL(Object Definition Language)设计方法：面向对象的数据库设计方法

计算机辅助设计：ORACLE Designer 2000；SYBASE PowerDesigner

3.数据库设计的基本步骤

6个阶段：需求分析-->概念结构设计-->逻辑结构设计-->物理结构设计-->数据库实施-->数据库运行和维护

需求分析和概念设计独立于任何数据库管理系统；逻辑设计和物理设计与选用的DBMS密切相关

(1)数据库设计的准备工作：选定参加设计的人

系统分析人员、数据库设计人员：数据库设计的核心人员，自始至终参与数据库设计

用户和数据库管理员：主要参加需求分析和数据库的运行维护

应用开发人员(程序员和操作员)：在系统实施阶段参与进来，负责编制程序和准备软硬件环境

1. 数据库设计的过程(六个阶段)

需求分析阶段：准确了解与分析用户需求(包括数据与处理)；最困难、最耗费时间的一步

概念结构设计阶段：整个数据库设计的关键；通过对用户需求进行综合、归纳与抽象，形成一个独立于具体DBMS的概念模型

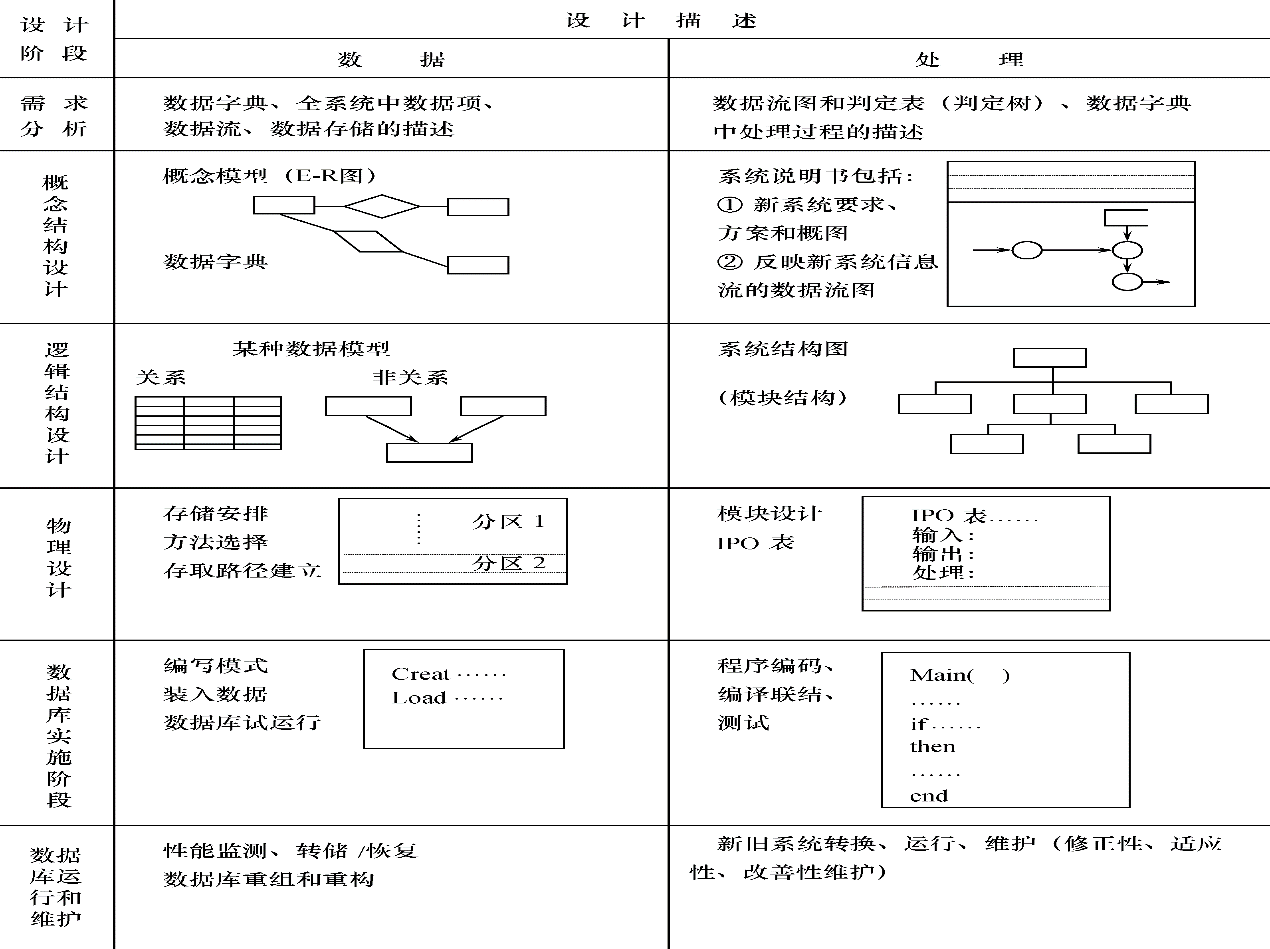
逻辑结构设计阶段L将概念结构转换为某个DBMS所支持的数据模型，对其进行优化

数据库物理设计阶段：为逻辑数据模型选取一个最适合应用环境的物理结构(包括存取结构和存取方法)

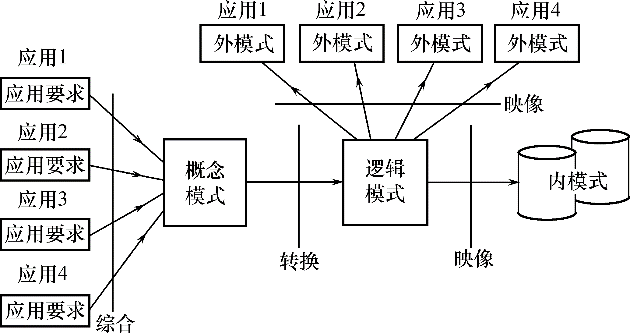
数据库实施阶段：运用DBMS提供的数据库语言(如SQL)及宿主语言，根据逻辑设计和物理设计的结果-->建立数据库；编制与调试应用程序；组织数据入库；进行试运行

数据库运行与维护阶段：数据库应用系统经过试运行后即可投入正常运行；在数据库系统运行过程中必须不断地对其进行评价、调整、修改

设计一个完善的数据库应用系统是上述六个阶段的不断反复

把数据库设计和对数据库中数据处理的设计精密结合起来；将这两个方面的需求分析、抽象、设计、实现在各个阶段同时进行，相互参照，相互补充，以完善两方面的设计

4，数据库设计过程中的各级模式

 数据库设计不同阶段行程单数据库各级模式

(二)需求分析

1.需求分析的任务

(1)需求分析的任务

详细调查现实世界要处理的对象(组织部门企业等)；充分了解源系统(手工系统或计算机系统)；明确用户的各种需求；确定新系统的功能；充分考虑今后可能得扩充和改变

(2)需求分析的重点

调查的重点是“数据”和“处理”，获得用户对数据库的要求：信息要求；处理要求；安全性与完整性要求

(3)需求分析的难点

确定用户最终需求：用户缺少计算机知识；设计人员缺少用户的专业知识

解决方法：设计人员必须不断深入地与用户进行交流

2.需求分析的方法

(1)调查需求

调查组织机构情况；调查各部门的业务活动；在熟悉业务活动的基础上，协助用户明确对新系统的各种要求；确定新系统的边界

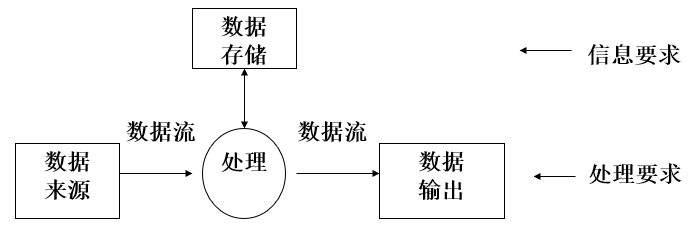
常用的调查方法：跟班作业；开调查会；请专人调查；询问；设计调查表请用户填写；查阅记录

结构化分析方法(Structured Analysis简称SA方法)：从最上层的系统组织结构入手；自定向下、逐层分解分析系统

(2)达成共识

(3)分析表达需求

a.首先把任何一个系统都抽象为：



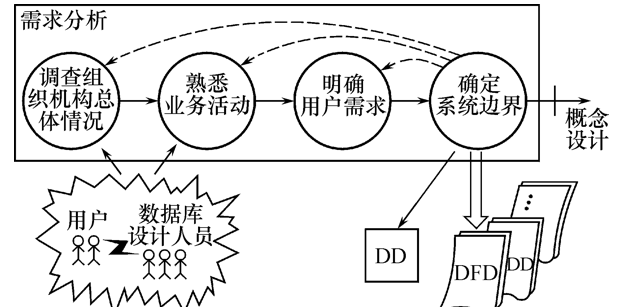
b.分解处理功能和数据：

分解处理功能：将处理功能的具体内容分解为若干子功能

分解数据：处理功能逐步分解同时，主机分解所用数据，形成若干层次的数据流图

表达方法：处理逻辑-->用判定表或判定树来表述；数据-->用数据字典来描述

c.将分析结果再次提交给用户，得到用户的认可



设计人员应充分考虑到可能得扩充和改变，是设计易于修改，系统易于扩充；必须用户用的参与

3.数据字典

数据字典是关于数据库中数据的描述，是元数据，而不是数据本身；数据字典在需求阶段建立，在数据库设计过程中不断修改、充实、完善

用途：进行详细的数据收集和数据分析所获得的主要结果

内容：数据项；数据结构；数据流；数据存储；处理过程

1. 数据项

数据项是不可再分的数据单位

对数据项的描述：数据项描述={数据项名，数据项含义说明，别名，数据类型，长度，取值范围，取值含义，与其他数据项的逻辑关系，数据项之间的联系}

(2)数据结构

数据结构反映了数据之间的组合关系；一个数据结构可以由若干个数据项组成，也可以由若干个数据结构组成，或由若干个数据项和数据结构混合组成；

对数据结构的描述：数据结构描述={数据结构名，含义说明，组成：{数据项或数据结构}}

(3)数据流

数据流是数据结构在系统内传输的路径

对数据流的描述：数据流描述={数据流名，说明，数据流来源，数据流去向，组成：{数据结构}，平均流量，高峰期流量}

(4)数据存储

数据存储是数据结构停留或保存的地方，也是数据流的来源和去向之一

对数据存储的描述：数据存储描述={数据存储名，说明，编号，组成：{数据结构}，数据量，存取频度，存取方式}

(5)处理过程

具体处理逻辑一般用判定表或判定树来描述

处理过程说明性信息的描述：处理过程描述={处理过程名，说明，输入：{数据流}，输出：{数据流}，处理：{简要说明}}

(6)数据字典举例

e.g.学生学籍管理子系统的数据字典

数据项：以“学号”为例

数据项：学号；含义说明：唯一标识每一个学生；别名：学生编号；类型：字符型；长度：8；取值范围：00000000~99999999；取值含义：前两位表别年级，后六位按顺序编号；与其他数据项的逻辑关系：

数据结构，以“学生”为例

“学生”是该系统中的一个核心数据结构；数据结构：学生；含义说明：是学籍管理子系统的主体数据结构，定义了一个学生的有关信息；组成：学号，姓名，性别，年龄，所在系，年级

数据流，“体检结果”可如下描述：

数据流：体检结果；说明：学生参加体格检查的最终结果；数据流来源：体检数据流去向：批准；组成：……；平均流量：……；高峰期流量：……

数据存储，“学生登记表”可如下描述：

数据存储，“学生登记表”可如下描述：

数据存储：学生登记表；说明：记录学生的基本情况；流入数据流：……；流出数据流：……；组成：……；数据量：每年3000张；存取方式：随机存取；

处理过程“分配宿舍”可如下描述：

处理过程：分配宿舍；说明：为所有新生分配学生宿舍；输入：学生，宿舍；输出：宿舍安排；处理：在新生报到后，为所有新生分配学生宿舍。要求同一间宿舍只能安排同一性别的学生，同一个学生只能安排在一个宿舍中。每个学生的居住面积不小于3平方米。安排新生宿舍其处理时间应不超过15分钟。

(三)概念结构设计

1.概念结构

概念设计是将需求分析得到的用户需求抽象为信息结构即概念模型的过程

概念结构式各种数据模型的共同基础，它比数据模型更独立于机器，更抽象，更加稳定

概念结构设计是整个数据库设计的关键

现实世界-->需求分析-->信息世界-->概念结构设计-->机器世界

(1)概念结构设计的特点：能真实、充分地反映现实世界；易于理解；易于更改；易于向关系、网状、层次等各种数据模型转换

(2)描述概念模型的工具

a.E-R模型；

b.扩展的E-R模型

b.1ISA联系：子类继承父类属性，也有自己的属性—分类属性；不相交约束和可重叠约束；完备性约束

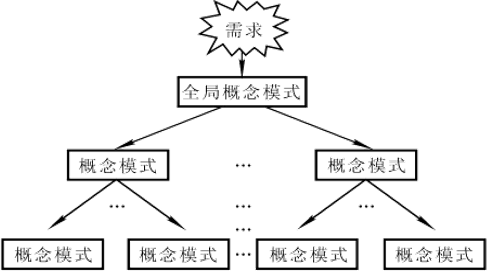
b.2基数约束

b.3 Part of约束

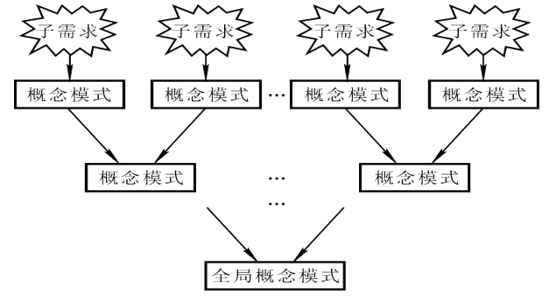
2.概念结构设计的方法和步骤

(1)设计概念结构的四类方法：

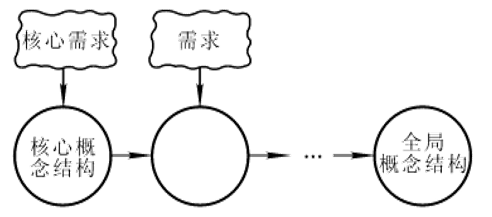
a.自顶向下：首先定义全局概念结构的框架，之后逐步细化



b.自底向上：首先定义各局部应用的概念结构，然后将他们集成起来得到全局概念结构

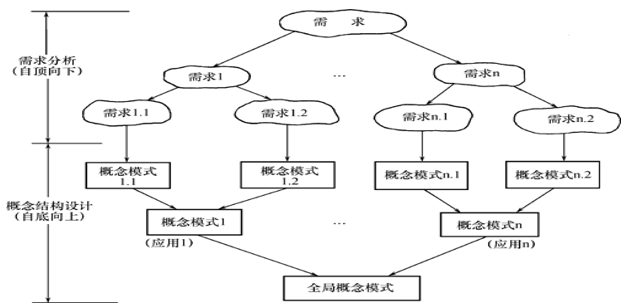


c.逐步扩张：首先心意最重要的核心概念结构，然后向外扩充，以滚雪球的方式逐步生成其他概念结构，直至总体概念结构

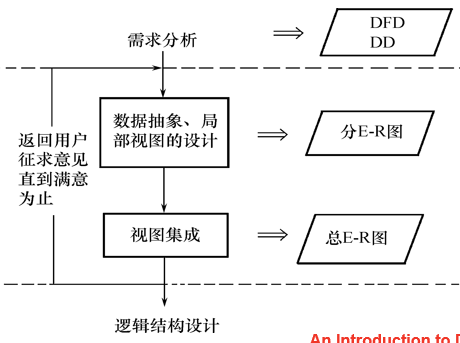


d.混合策略：将自顶向下和自底向上相结合，用自顶向下策略设计一个全局概念结构的框架，以它为骨架集成由西地向下策略中设计的各局部概念结构

e.常用策略：自顶向下地进行需求分析，自底向上的进行概念结构设计



自底向上设计概念结构的步骤：抽象数据并设计局部视图-->集成局部视图，得到全局概念结构

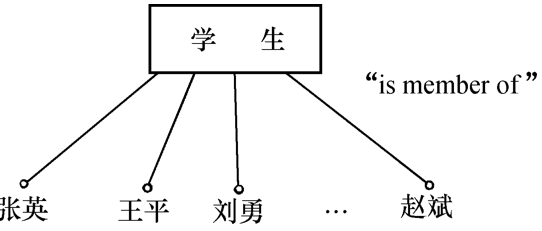


3.数据抽象和局部视图设计

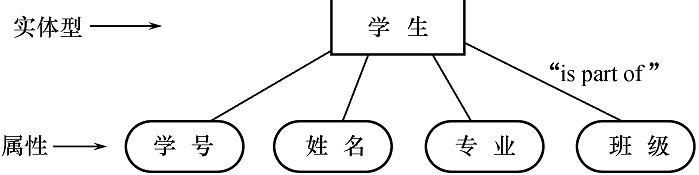
数据抽象:抽象是对实际的人、物、事和概念中抽取所关心的共同特性，忽略非本质的细节，并把这些特性用各种概念精确的加以描述。概念结构是对现实世界的一种抽象

(1)三种常用抽象：

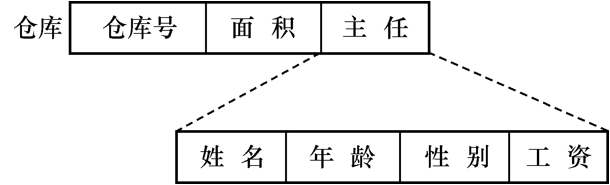
分类：定义某一类概念作为现实世界中一组对象的类型；抽象了对象值和型之间的“is member of”的语义



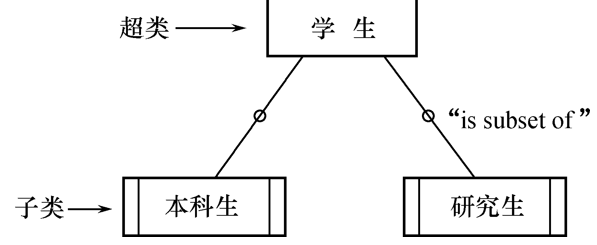
聚集：定义某一类型的组成成分，抽象了对象内部类型和成分之间“is part of”的语义



复杂的聚集，某一类型的成分仍是一个聚集



概括：定义类型之间的一种子集联系，抽象了类型之间的“is subset of”的语义；有继承性

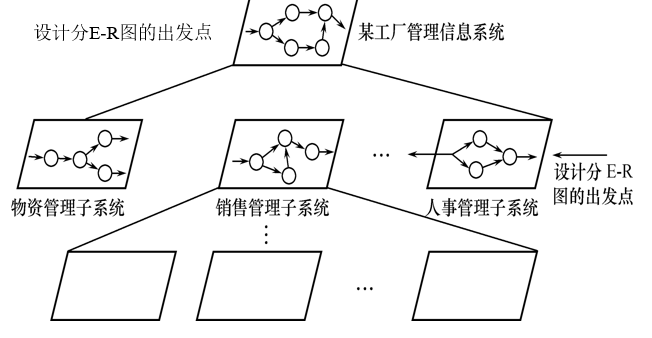


(2)局部视图设计

设计分E-R图的步骤：选择局部应用，逐一设计分E-R图

a.选择局部应用

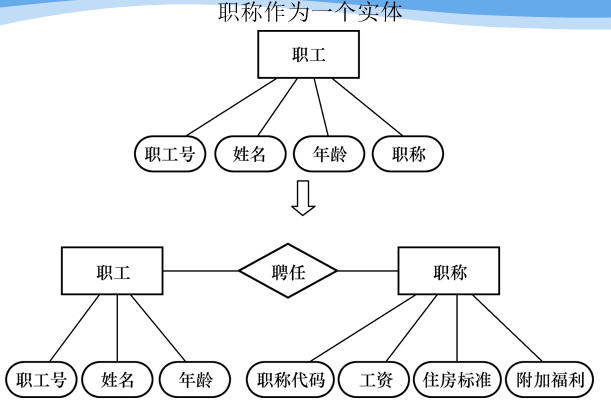
在多层的数据流图中选择一个适当层次的数据流图，作为设计分E-R图的出发点，通常以中层数据流图作为设计分E-R图的依据。

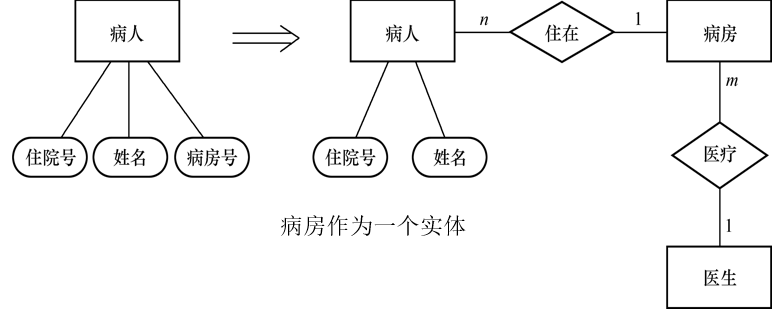


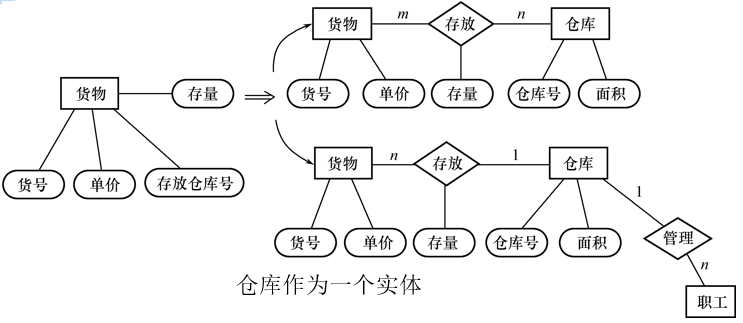
b.逐一设计分E-R图

任务：将各局部应用设计的数据分别从数据字典中抽取出来；参照数据流图，标定个局部应用中的实体、实体的属性、标识实体的码；确定实体之间的联系及其类型(1:1,1:n.m:n)

两条准则：属性不能再具有需要描述的性质，即属性必须是不可分的数据项，不能再由另一些属性组成；属性不能再与其他实体具有联系，联系只发生在实体之间





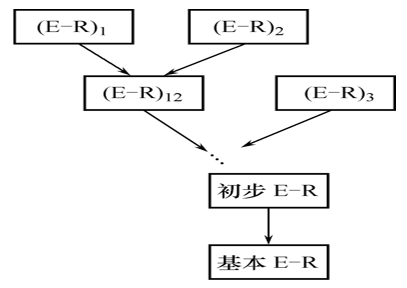
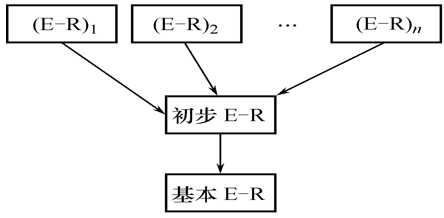


4.视图的集成

a.视图集成的两种方式

各个局部视图即分E-R图建立好后，还需要对他们惊醒合并，集成为一个成体的数据概念结构即总E-R图

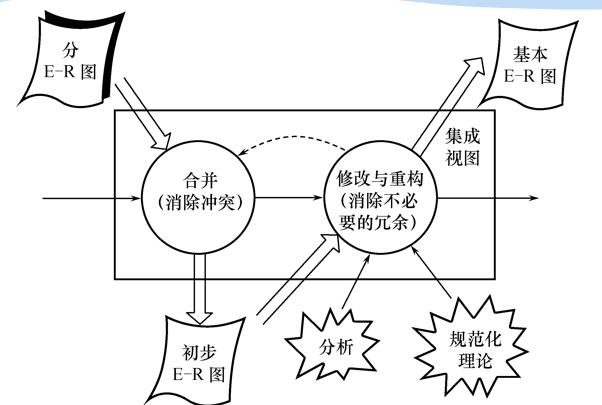
a.1多个分E-R图一次集成：一次集成多个分E-R图，通常用于局部视图比较简单的情况(下左图)



a.2逐步集成：用累加的方式一次集成两个分E-R图

上右图

集成局部E-R图的步骤：合并；修改与重构



b.合并分E-R图，生成初步E-R图

各分E-R图存在冲突：各个分E-R图之间必定会有很多不一致的地方

合并分E-R图的主要工作和关键：合理消除各分E-R图的冲突

冲突的种类：属性冲突、命名冲突、结构冲突

b.1两类属性冲突

属性域冲突：属性值的类型；取值范围；取值集合不同

属性取值单位冲突

b.2两类命名冲突

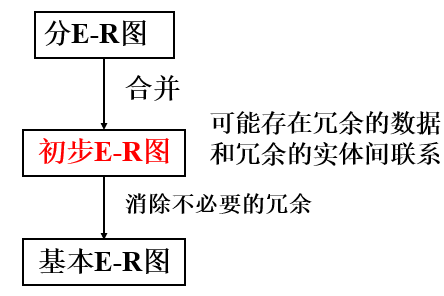
同名异义:不同意义的对象在不同的局部应用中具有相同的名字

异名同义(一义多名)：同一意义的对象在不同的局部应用中具有不同的名字

b.3三类结构冲突

同一对象在不同应用中具有不同的抽象

同一实体在不同分E-R图中所包含的属性个数和属性排列次序不完全相同

 实体之间的联系在不同局部视图中呈现不同的类型

c.消除不必要的冗余，设计基本E-R图

基本任务：消除不必要的冗余，设计基本E-R图

c.1冗余

冗余的数据：可由基本数据导出的数据

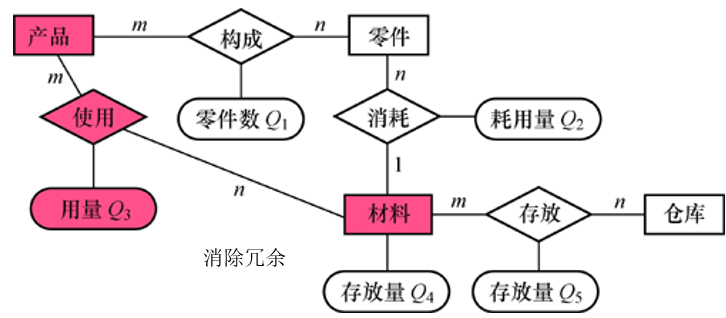
冗余的联系：可由其他联系导出的联系

冗余数据和冗余联系容易破坏数据库的完整性，给数据库维护增加困难

消除不必要的冗余后的初步E-R图称为基本E-R图

c.2消除冗余的方法

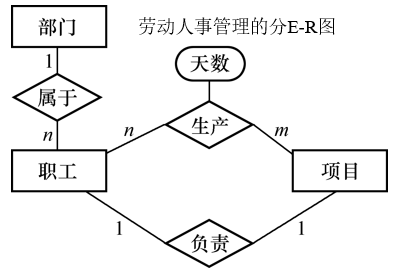
分析方法：以数据字典和数据流图为依据；根据数据字典中关于数据项之间的逻辑关系



效率vs冗余信息：根据用户的整体需求确定

若认为的保留了一些冗余数据，则应把数据字典中数据关联的说明作为完整性约束条件：Q4=∑Q5，一旦Q5修改后就应当触发完整性检查，对Q4进行修改

规范化理论：函数依赖的概念提供了消除冗余联系的形式化工具

方法：

c.2.1确定分E-R图实体之间的数据依赖，并用实体码之间的函数依赖表示

右图中，部门和职工之间一对多的联系可表示为：职工号→部门号 职工和产品之间多对多的联系可表示为：职工号，产品号)→工作天数

得到函数依赖集FL

c.2.2求FL的最小覆盖GL，差集为D=FL-GL

逐一考察D中的函数依赖，确定是否是冗余的联系，如果是，就把他去掉

冗余的联系一定在D中，而D中的联系不一定是冗余的

当实体之间存在多种联系时，要将实体之间的联系在形式上加以区分

d.验证整体概念结构

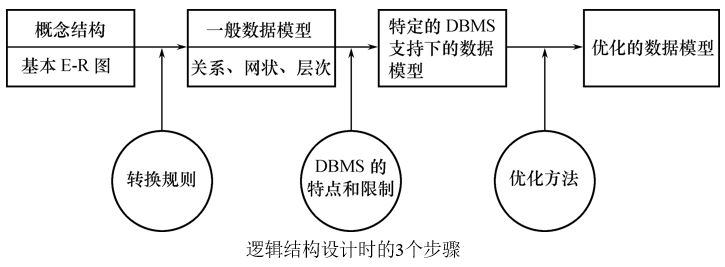
视图继承后形成一个整体的数据库概念结构，对该整体概念结构还必须惊醒进一步验证，确保它满足下列条件：整体概念结构内部具有一致性，不存在互相矛盾的表达；整体概念结构能准确的反应原来的每个视图结构，包括属性、实体、实体之间的联系；整体概念结构设计能满足需求分析阶段所确定的所有要求

整体概念结构最终还应该提交给用户，征求用户和有关人员的意见，进行评审、修改和优化，然后把它确定下来，作为数据库的概念结构，作为进一步设计数据库的依据

(四)逻辑结构设计

逻辑结构设计的任务：把概念结构设计阶段设计好的基本E-R图转换为与选用DBMS产品所支持的数据模型相符合的逻辑结构

逻辑结构设计的步骤：将概念结构转化为一般的关系、网状、层次模型；将转换来的关系、网状、层次模型向特定DBMS支持下的数据模型转换；对数据模型进行优化



1.E-R图向关系模型的转换

(1)转换内容

E-R图向关系模型的转换要解决的问题：如何将实体型和实体间的联系转换为关系模式；如何确定这些关系模式的属性和码

转换的内容：将E-R图转换为关系模型，将实体、实体的属性、实体之间的联系转换为关系模式

(2)转换原则

实体之间的联系有不同的情况

1. 一个1:1联系可以转换为一个独立的关系模式，也可以与任意一段对应的关系模式合并：转换为一个独立的关系模式，与某一端实体对应的关系模式合并
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：转换为一个独立的关系模式，与n端对应的关系模式合并
3. 一个m：n联系转换为一个关系模式

e.g.选修是一个m：n关系，可以将之转换为如下关系模式：选修(学号，课程号，成绩)，其中学号和课程号位关系的组合码

d．三个或三个以上实体间的一个多远关系转换为一个关系模式

e.g.讲授是一个三元关系，可以将它转换为如下关系模式：讲授(课程号，职工号，书号)

e．具有相同码的关系模式可以合并。目的：减少系统中的关系个数。合并方法：将其中一个关系模式的全部属性加入到另一个关系模式中，让后去掉其中的同义属性(可能同名也可能不同名)，并适当调整属性的次序

注：从理论上讲，1:1联系可以与任意一段对应的关系模式合并。但是在一些情况下，与不同的关系模式合并效率会大不一样，究竟该与那一段的关系模式合并需要依据应用的具体情况而定。由于连接操作时最费时的操作，所以一般应尽量减少连接操作作为目标。例如：如果经常要查询某个班级的班主任姓名，则将管理联系与教师关系合并更好些

2．数据模型的优化

得到初步数据模型后，还应该适当地自改、调整数据模型的结构，以进一步提高数据库应用洗头膏的性能，这就是数据模型的优化。关系数据模型的优化通常以规范化理论为指导

优化数据模型的方法：

(1)确定数据依赖

按需求分析阶段所得的语义，分别写出每个关系模式内部各属性之间的数据依赖以及不同关系模式属性之间的数据依赖

(2)消除冗余的联系

对于各个关系模式之间的数据依赖进行极小化处理，消除冗余的联系

(3)确定所属范式

按照数据依赖的理论对关系模式注意进行分析；考察是否存在部分函数依赖、传递函数依赖、多值依赖等；确定个关系模式分别属于第几范式

(4)按照需求分析阶段得到的各种应用对数据处理的要求，分析对于这样的应用环境这些模式是否合适，确定是否要对它们进行分解

注：并不是规范化程度越高越好，一般的，第三范式就够了

(5)按照需求分析阶段得到的各种应用对数据处理的要求，对关系模式进行必要的分解，以提高数据操作的效率和存储空间的利用率

常用的分解方法有：

水平分解(按行分解)：把基本关系的元组分为若干个子集合，定义每个子集合为一个子关系，以提高系统的效率。适用范围-->满足“80/20原则的应用”l；并发事务经常存取不相交的数据

垂直分解(按列分解)：把关系模式R的属性分解为若干个子集合，形成若干子关系模式。适用范围-->取决于分解后R上的所有事务的总效率是否得到提高

3.设计用户子模式

定义用户外模式时应注重：使用符合用户习惯的别名；针对不同级别的用户定义不同的VIEW，以满足系统对安全性的要求；简化用户对系统的使用

e.g. 关系模式产品(产品号，产品名，规格，单价，生产车间，生产负责人，产品成本，产品合格率，质量等级)，可以在产品关系上建立两个视图：

为一般顾客建立视图：产品1(产品号，产品名，规格，单价)

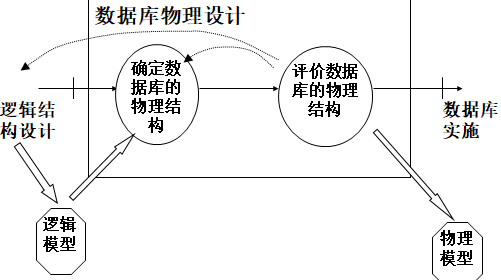
为产品销售部门建立视图：产品2(产品号，产品名，规格，单价，车间，生产负责人)

顾客视图中只包含允许顾客查询的属性；销售部门视图中只包含允许销售部门查询的属性；生产领导部门则可以查询全部产品数据；可以防止用户非法访问不允许他们查询的数据，保证系统的安全性

(五)数据库的物理设计

数据库在物理设备上的存储结构与存取方法称为数据库的物理结构，它依赖于选定的数据库管理系统为一个给定的逻辑数据模型选取一个最适合应用黄静的物理结构的过程，就是数据库的物理设计。建立索引在物理结构的范围

步骤：确定数据库的物理结构，在关系数据库中主要指存取方法和存储结构，对物理结构进行评价，重点是时间和空间效率。如果评价结果吗满足原设计要求，可以进入到物理实施阶段，否则就需要重新设计或者修改物理结构，有时甚至要返回逻辑设计阶段修改数据模型



1数据库物理设计的内容和方法

设计物理数据库结构的准备工作：对要运行的事务进行详细分析，获得选择物理数据库设计所需要的参数；充分了解所用RDBMS的内部特征，特别系统提供的存取方法和存取结构

(1)选择物理数据库设计所需要的参数

数据更新事务：被更新的关系，每个关系上的更新操作条件所涉及的属性，修改操作要改变的属性值

每个事务在各关系上运行的频率和性能要求

(2)关系数据库物理设计的内容

为关系模式选择存取方法(建立存取路径)，设计关系、索引等数据库文件的物理存储结构；

2.关系模式存取方法选择

数据库系统是多用户共享的系统，对用一个关系要建立多条存取路径才能满足多用户的多种应用要求

物理设计的任务之一就是确定选择那些存取方法，即建立哪些存取路径

(1)DBMS常用的存取方法：

索引方法：目前主要是B+树索引方法；经典存取方法(使用最为普遍)；聚簇方法；HASH方法

索引存取方法的选择

根据应用要求确定，对哪些属性列建立索引，对哪些属性列建立组合索引，对哪些索引要设计为唯一索引

(2)选择索引存取方法的一般规则

一个(或一组)属性经常在查询条件中出现，则考虑在这个(或这组)属性上建立索引(或组合索引)

如果一个属性经常作为最大值和最小值等聚集函数的参数，则考虑在这个属性上建立索引

如果一个(或一组)属性经常在连接操作的连接条件中出现，则考虑在这个(或这组)属性上建立索引

关系上定义的索引数过多会带来较多的额外开销: 维护索引的开销； 查找索引的开销

(3)聚簇存取方法的选择

聚簇：为了提高某个属性(或属性组)的查询速度，把这个或这些属性(称为聚簇码)上具有相同值的元组集中存放在连续的物理块称为聚簇

a.聚簇的用途

a.1.大大提高按聚簇码进行查询的效率

例：假设学生关系按所在系建有索引，现在要查询信息系的所有学生名单。

信息系的500名学生分布在500个不同的物理块上时，至少要执行500次I/O操作

如果将同一系的学生元组集中存放，则每读一个物理块可得到多个满足查询条件的元组，从而显著地减少了访问磁盘的次数

a.2.节省存储空间

聚簇以后，聚簇码相同的元组集中在一起了，因而聚簇码值不必在每个元组中重复存储，只要在一组中存一次就行了

b. 聚簇的局限性

b.1.聚簇只能提高某些特定应用的性能

b.2.建立与维护聚簇的开销相当大

对已有关系建立聚簇，将导致关系中元组移动其物理存储位置，并使此关系上原有的索引无效，必须重建

当一个元组的聚簇码改变时，该元组的存储位置也要做相应移动

c. 聚簇的适用范围

c.1.既适用于单个关系独立聚簇，也适用于多个关系组合聚簇

例：假设用户经常要按系别查询学生成绩单，这一查询涉及学生关系和选修关系的连接操作，即需要按学号连接这两个关系，为提高连接操作的效率，可以把具有相同学号值的学生元组和选修元组在物理上聚簇在一起。这就相当于把多个关系按“预连接”的形式存放，从而大大提高连接操作的效率。

c.2.当通过聚簇码进行访问或连接是该关系的主要应用，与聚簇码无关的其他访问很少或者是次要的时，可以使用聚簇。

尤其当SQL语句中包含有与聚簇码有关的ORDER BY，GROUP BY，UNION，DISTINCT等子句或短语时，使用聚簇特别有利，可以省去对结果集的排序操作

d.设计候选聚簇

对经常在一起进行连接操作的关系可以建立聚簇

如果一个关系的一组属性经常出现在相等比较条件中，则该单个关系可建立聚簇

如果一个关系的一个(或一组)属性上的值重复率很高，则此单个关系可建立聚簇。即对应每个聚簇码值的平均元组数不太少。太少了，聚簇的效果不明显

e. 优化聚簇设计

从聚簇中删除经常进行全表扫描的关系；

从聚簇中删除更新操作远多于连接操作的关系；

不同的聚簇中可能包含相同的关系，一个关系可以在某一个聚簇中，但不能同时加入多个聚簇-->从这多个聚簇方案(包括不建立聚簇)中选择一个较优的，即在这个聚簇上运行各种事务的总代价最小

(4)HASH存取方法的选择

选择HASH存取方法的规则，当一个关系满足下列两个条件时，可以选择HASH存取方法：该关系的属性主要出现在等值连接条件中或主要出现在相等比较选择条件中；该关系的大小可预知，而且不变；或该关系的大小动态改变，但所选用的DBMS提供了动态HASH存取方法

3.确定数据库的存储结构

确定数据库物理结构的内容

确定数据的存放位置和存储结构：关系、索引、聚簇、日志、备份

确定系统配置

(1)确定数据的存放位置

a.确定数据存放位置和存储结构的因素：存取时间、存储空间利用率、维护代价

这三个方面常常是相互矛盾的，例：消除一切冗余数据虽能够节约存储空间和减少维护代价，但往往会导致检索代价的增加

必须进行权衡，选择一个折中方案

b.基本原则

根据应用情况将易变部分与稳定部分分开存放；

存取频率较高部分与存取频率较低部分，分开存放

例：数据库数据备份、日志文件备份等由于只在故障恢复时才使用，而且数据量很大，可以考虑存放在磁带上

如果计算机有多个磁盘或磁盘阵列 ，可以考虑将表和索引分别放在不同的磁盘上，在查询时，由于磁盘驱动器并行工作，可以提高物理I/O读写的效率

例(续)：

可以将比较大的表分别放在两个磁盘上，以加快存取速度，这在多用户环境下特别有效

可以将日志文件与数据库对象(表、索引等)放在不同的磁盘以改进系统的性能

(2)确定系统配置

DBMS产品一般都提供了一些存储分配参数：同时使用数据库的用户数；同时打开的数据库对象数；内存分配参数；使用的缓冲区长度、个数；存储分配参数……

4.评价物理结构

(1)评价内容:对数据库物理设计过程中产生的多种方案进行细致的评价，从中选择一个较优的方案作为数据库的物理结构

(2)评价方法(完全依赖于所选用的DBMS )

定量估算各种方案：存储空间；存取时间；维护代价

对估算结果进行权衡、比较，选择出一个较优的合理的物理结构

如果该结构不符合用户需求，则需要修改设计

(六)数据库实施和维护

1.数据的载入和应用程序的调试

(1)数据的载入

数据库结构建立好后，就可以向数据库中装载数据了。组织数据入库是数据库实施阶段最主要的工作。

数据装载方法：人工方法；计算机辅助数据入库

(2)应用程序的编码和测试

数据库应用程序的设计应该与数据设计并行进行

在组织数据入库的同时还要调试应用程序

2.数据库的试运行

在原有系统的数据有一小部分已输入数据库后，就可以开始对数据库系统进行联合调试，称为数据库的试运行

数据库试运行主要工作包括：

(1)功能测试

实际运行数据库应用程序，执行对数据库的各种操作，测试应用程序的功能是否满足设计要求

如果不满足，对应用程序部分则要修改、调整，直到达到设计要求

(2)性能测试

测量系统的性能指标，分析是否达到设计目标

如果测试的结果与设计目标不符，则要返回物理设计阶段，重新调整物理结构，修改系统参数，某些情况下甚至要返回逻辑设计阶段，修改逻辑结构

(3) 强调两点：

a.分期分批组织数据入库

重新设计物理结构甚至逻辑结构，会导致数据重新入库。

由于数据入库工作量实在太大，费时、费力，所以应分期分批地组织数据入库

先输入小批量数据供调试用：待试运行基本合格后再大批量输入数据；逐步增加数据量，逐步完成运行评价

b. 数据库的转储和恢复

在数据库试运行阶段，系统还不稳定，硬、软件故障随时都可能发生；系统的操作人员对新系统还不熟悉，误操作也不可避免；因此必须做好数据库的转储和恢复工作，尽量减少对数据库的破坏。

3.数据库的运行和维护

数据库试运行合格后，数据库即可投入正式运行。

数据库投入运行标志着开发任务的基本完成和维护工作的开始

对数据库设计进行评价、调整、修改等维护工作是一个长期的任务，也是设计工作的继续和提高-->应用环境在不断变化；数据库运行过程中物理存储会不断变化

在数据库运行阶段，对数据库经常性的维护工作主要是由DBA完成的，包括：数据库的转储和恢复；数据库的安全性、完整性控制；数据库性能的监督、分析和改进；数据库的重组织和重构造

a.数据库的重组织

重组织的形式：全部重组织；部分重组织-->只对频繁增、删的表进行重组织

重组织的目标：提高系统性能

重组织的工作：

按原设计要求-->新安排存储位置；回收垃圾；减少指针链

数据库的重组织不会改变原设计的数据逻辑结构和物理结构

b. 数据库重构造

根据新环境调整数据库的模式和内模式：增加新的数据项；改变数据项的类型；改变数据库的容量；增加或删除索引；修改完整性约束条件

**八、数据库编程**

(一)嵌入式SQL

SQL语言提供了两种不同的使用方式：交互式；嵌入式

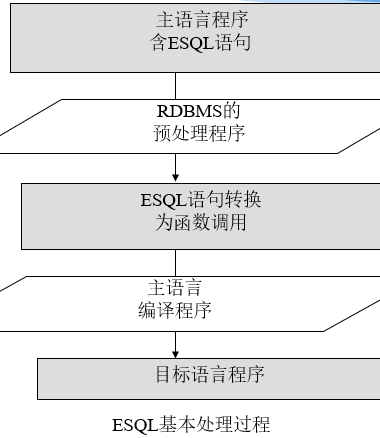
为什么要引入嵌入式SQL：SQL语言是非过程性语言；事务处理应用需要高级语言

这两种方式细节上有差别，在程序设计的环境下，SQL语句要做某些必要的扩充

1.嵌入式SQL的处理过程

主语言：嵌入式SQL是将SQL语句嵌入程序设计语言中，被嵌入的程序设计语言，如C、C++、Java，称为宿主语言，简称主语言

处理过程：预编译方法



为了区分SQL语句与主语言语句， 所有SQL语句必须加前缀EXEC SQL，以(;)结束:

EXEC SQL <SQL语句>;

2.嵌入式SQL语句与主语言之间的通信

将SQL嵌入到高级语言中混合编程，程序中会含有两种不同计算模型的语句：

SQL语句：描述性的面向集合的语句；负责操纵数据库

高级语言语句：过程性的面向记录的语句；负责控制程序流程

数据库公国单元和源程序公国单元之间的通信：

SQL通信区：向主语言传递SQL语句的执行状态信息；使主语言能够据此控制程序流程

主变量：主语言向SQL语句提供参数；将SQL语句查询数据库的结果交主语言进一步处理

游标：解决几何形操作语言与过程性操作语言的不匹配

(1)SQL通信区(SQL communication area)

SQLCA是一个数据结构

a.SQLCA的用途：

SQLCA语句执行后，RDBMS反馈给应用程序信息用来描述系统当前工作状态和运行环境，这些信息将行到SQL通信区SQLCA中， 应用程序从SQLCA中取出这些状态信息，据此决定接下来执行的语句

b.SQLCA使用方法

定义SQLCA：用EXEC SQL INCLUDE SQLCA定义

使用SQLCA：SQLCA中有一个存放每次执行SQL语句后返回代码的变量SQLCODE

(2)主变量

嵌入式SQL语句中可以使用主语言的程序变量来输入或者输出数据

在SQL语句中使用的主语言程序变量简称为主变量(Host Variable)

a.主变量的类型

输入主变量；输出主变量；一个主变量有可能既是输入主变量又是输出主变量

指示变量：一个主变量可以附带一个指示变量

SQL为系统定义的全局变量，供应用程序与DBMS通信使用：使用时在应用程序中加：EXEC SQL INCLUDE SQLCA，不必再有用户说明；SQLCA有一个分量叫SQLCODE，可以表示为：SQLCA.SQLCADE；每执行一条SQL语句，都要返回一个SQLCODE的值(0表示执行成功，正数表示已经执行但有异常，100表示无数据可以取，负数表示出错或未执行等)

b.在SQL语句中使用主变量和指示变量的方法

b.1说明主变量和指示变量

BEGIN DECLARE SECTION

……

……(说明主变量和指示变量)

END DECLARE SECTION

Exec sql begin declare section

char sno[7];

char cno[7];

Float GRADE;

Shot Grade1; 指示变量，与GRADE连用才有意义

Exec sql end declare section

b.2使用主变量

说明之后的主变量可以在SQL语句中的任何一个能够使用表达式的地方出现；为了与数据库对象名(表名、视图名、列名)区别，SQL语句中的主变量名前要加冒号(:)作为标志

b.3使用指示变量

指示变量前也必须要加冒号标志，必须紧跟在所指主变量之后

c.在SQL语句之外(主语言中)使用主变量和指示变量的方法

可以直接应用不加冒号

d.嵌入式SQL的可执行语句

包括DDL、QL、DML，DCL；与SQL基本一致，增加少许语法成分；还包括进入数据库的语句connect和控制事务结束的语句

比如：EXEC SQL CONNECT :uid IDENTIFIED BY :pwd

:uid 宿主变量,用户标示

:pwd宿主变量,用户口令.

又如: EXEC SQL INSERT INTO SC(SNO,CNO,GRADE)

VALUES(:SNO,:CNO,:GRADE);

(3)游标

为什么使用游标：SQL语言和主语句具有不同的数据处理方式；SQL语言是面向集合的，一条SQL语句原则上可以产生或处理多条记录；主语言是面向记录的，一组主变量一次只能存放一条记录；进使用主变量并不能完全满足SQL语句向应用程序输出数据的要求；嵌入式SQL引入了游标的概念，用来协调这两种不同的处理方式

游标：

游标是系统为用户开设的一个数据缓冲区，存放SQL语句的执行结果；每个游标区都有一个名字；用户可以用SQL语句逐一从游标中获取记录，并赋给主变量，交由主语言进一步处理

说明游标语句：exec sql declare <游标名> cursor for

Select …

From …

Where …;

打开游标语句： exec sql open <游标名> ;

取数语句：EXEC SQL FETCH <游标名> INTO :hv1,:hv2,…;

一般用在循环语句里,当无数可取时,sqlcode取100.

关闭游标语句：EXEC SQL CLOSE <游标名>;使用示例:p136

success:取数成功与否

(4)建立和关闭数据库连接

a.建立数据库连接

EXEC SQL CONNECT TO target[AS connection-name][USER user-name];

target是要连接的数据库服务器

常见的服务器表示串，如<dbname>@<hostname>:<port>

包含服务器标识的SQL串常量

DEFAULT

Connect-name是可选的连接名，连接必须是一个有效的标识符，在整个程序内只有一个连接时可以不指定连接名

b.关闭数据连接

EXEC SQL DISCONNECT [connection]

c.程序运行过程中可以修改当前连接

EXEC SQL SET CONNECTION connection-name|DEFAULT

(5)程序实例

[例1]依次检查某个系的学生记录，交互式更新某些学生年龄。

EXEC SQL BEGIN DECLARE SECTION; /\*主变量说明开始\*/

char deptname[64];

char HSno[64];

char HSname[64];

char HSsex[64];

int HSage;

int NEWAGE;

EXEC SQL END DECLARE SECTION; /\*主变量说明结束\*/

long SQLCODE;

EXEC SQL INCLUDE sqlca; /\*定义SQL通信区\*/

int main(void) { /\*C语言主程序开始\*/

int count = 0;

char yn; /\*变量yn代表yes或no\*/

printf("Please choose the department name(CS/MA/IS): ");

scanf("%s", deptname); /\*为主变量deptname赋值\*/

EXEC SQL CONNECT TO TEST@localhost:54321 USER

"SYSTEM" /"MANAGER"; /\*连接数据库TEST\*/

EXEC SQL DECLARE SX CURSOR FOR /\*定义游标\*/

SELECT Sno, Sname, Ssex, Sage /\*SX对应语句的执行结果\*/

FROM Student

WHERE SDept = :deptname;

EXEC SQL OPEN SX; /\*打开游标SX便指向查询结果的第一行\*/

for ( ; ; ) { /\*用循环结构逐条处理结果集中的记录\*/

EXEC SQL FETCH SX INTO :HSno, :HSname, :HSsex,:HSage;

/\*推进游标，将当前数据放入主变量\*/

if (sqlca.sqlcode != 0) /\* sqlcode != 0,表示操作不成功\*/

break; /\*利用SQLCA中的状态信息决定何时退出循环\*/

if(count++ == 0) /\*如果是第一行的话，先打出行头\*/

printf("\n%-10s %-20s %-10s %-10s\n", "Sno", "Sname", "Ssex", "Sage");

printf("%-10s %-20s %-10s %-10d\n", HSno, HSname, HSsex, HSage);

/\*打印查询结果\*/

printf("UPDATE AGE(y/n)?"); /\*询问用户是否要更新该学生的年龄\*/

do{

scanf("%c",&yn);

}while(yn != 'N' && yn != 'n' && yn != 'Y' && yn != 'y');

if (yn == 'y' || yn == 'Y'){ /\*如果选择更新操作\*/

printf("INPUT NEW AGE:");

scanf("%d",&NEWAGE); /\*用户输入新年龄到主变量中\*/

EXEC SQL UPDATE Student /\*嵌入式SQL\*/

SET Sage = :NEWAGE

WHERE CURRENT OF SX ;

} /\*对当前游标指向的学生年龄进行更新\*/

}

EXEC SQL CLOSE SX; /\*关闭游标SX不再和查询结果对应\*/

EXEC SQL COMMIT WORK; /\*提交更新\*/

EXEC SQL DISCONNECT TEST; /\*断开数据库连接\*/

}

3.不使用游标的SQL语句

不使用游标的SQL语句的种类：说明性语句；数据定义语句；数据控制语句；查询结果为单记录的SELECT语句；非CURRENT形式的增删改语句

(1)查询结果为单记录的SELECT语句

这类语句不需要使用游标，只需要用INTO子句指定存放查询结果的主变量

[例2] 根据学生号码查询学生信息。假设已经把要查询的学生的学号赋给了主变量givensno

EXEC SQL SELECT Sno，Sname，Ssex，Sage，Sdept

INTO **:**Hsno，**:**Hname，**:**Hsex，**:**Hage，**:**Hdept

FROM Student

WHERE Sno=**:**givensno；

INTO子句、WHERE子句和HAVING短语的条件表达式中均可以使用主变量

查询返回的记录中，可能某些列为空值NULL

如果查询结果实际上并不是单条记录，而是多条记录，则策划给你续出错，RDBMS会在SQLCA中返回错误信息

[例3] 查询某个学生选修某门课程的成绩。假设已经把将要查询的学生的学号赋给了主变量givensno，将课程号赋给了主变量givencno。

EXEC SQL SELECT Sno，Cno，Grade

INTO :Hsno，:Hcno，:Hgrade:Gradeid /\*指示变量Gradeid\*/

FROM SC

WHERE Sno=:givensno AND Cno=:givencno；

如果Gradeid < 0，不论Hgrade为何值，均认为该学生成绩为空值。

(2)非CURRENT形式的增删改语句

在UPDATE的SET子句和WHERE子句中可以使用主变量，SET子句还可以使用指示变量

[例4] 修改某个学生选修1号课程的成绩。

EXEC SQL UPDATE SC

SET Grade=:newgrade /\*修改的成绩已赋给主变量\*/

WHERE Sno=:givensno； /\*学号赋给主变量givensno\*/

[例5] 将计算机系全体学生年龄置NULL值。

Sageid=-1；

EXEC SQL UPDATE Student

SET Sage=:Raise :Sageid

WHERE Sdept= ‘CS’；

将指示变量Sageid赋一个负值后，无论主变量Raise为何值，RDBMS都会将CS系所有学生的年龄置空值 。

等价于：

EXEC SQL UPDATE Student

SET Sage=NULL

WHERE Sdept= 'CS'；

[例6] 某个学生退学了，现要将有关他的所有选课记录删除掉。假设该学生的姓名已赋给主变量stdname。

EXEC SQL DELETE

FROM SC

WHERE Sno=

(SELECT Sno

FROM Student

WHERE Sname=:stdname)；

[例7] 某个学生新选修了某门课程，将有关记录插入SC表中。假设插入的学号已赋给主变量stdno，课程号已赋给主变量couno。

gradeid=-1； /\*用作指示变量，赋为负值\*/

EXEC SQL INSERT

INTO SC(Sno，Cno，Grade)

VALUES(:stdno，:couno，:gr :gradeid)；

由于该学生刚选修课程，成绩应为空，所以要把指示变量赋为负值

4.使用游标的SQL语句

必须使用游标的SQL语句：查询结果为多条记录的SELECT语句；CURRENT形式的UPDATE语句；CURRENT形式的DELETE语句

(1) 查询结果为多条记录的SELECT语句

使用游标的步骤：说明游标，打开游标，推进游标并读取当前记录，关闭游标

a.说明游标

使用DECLARE语句，语句格式：EXEC SQL DECLARE <游标名> CURSOR

FOR <SELECT语句>;

功能：是一条说明性语句，这是DBMS并不执行SELECT指定的查询操作

b.打开游标

使用OPEN语句，语句格式：EXEC SQL OPEN<游标名>

功能：打开游标实际上是执行相应的SELECT语句，把所有满足查询条件的记录从指定表取到缓冲区中，这是游标处于活动状态，指针指向查询结果集中的第一条记录

c.推进游标并读取当前记录

使用FETCH语句，语句格式：

EXEC SQL FETCH [[NEXT|PRIOR|FIRST|LAST] FROM] <游标名>

INTO <主变量>[<指示变量>][,<主变量>[<指示变量>]]...;

功能：指定方向推动游标指针，然后将缓冲区中的当前记录取出来送至主变量供主语言进一步处理

NEXT|PRIOR|FIRST|LAST：指定推动游标指针的方式

NEXT：向前推进一条记录 PRIOR：向回退一条记录

FIRST：推向第一条记录 LAST：推向最后一条记录

缺省值为NEXT

d.关闭游标

使用CLOSE语句，语句格式：EXEC SQL CLOSE <游标名>;

功能：关闭游标，释放结果集占用的缓冲区及其他资源

说明：游标被关闭后，就不再和原来的查询结果集相联系；被关闭的游标可以再次被打开，与新的查询结果相联系

(2)CURRENT形式的UPDATE和DELETE语句

CURRENT形式的UPDATE语句和DELETE语句的用途：面向集合的操作；一次修改或删除所有满足条件的记录

如果只想修改或删除其中某个记录：用带游标的SELECT语句查出所有满足条件的记录；从中进一步找出要修改或删除的记录；用CURRENT形式的UPDATE语句和DELETE语句修改或删除之

UPDATE语句和DELETE语句中的子句：

WHERE CURRENT OF <游标名> //表示修改或删除的是最近一次取出的记录，即游标指针指向的记录

不能使用CURRENT形式的UPDATE语句和DELETE语句：当游标定义中的SELECT语句带有UNION或ORDER BY子句；该SELECT语句相当于定义了一个不可更新的视图

5.动态SQL

静态嵌入式SQL：静态嵌入式SQL语句能够满足一般要求 ；无法满足要到执行时才能够确定要提交的SQL语句

动态嵌入式SQL：允许在程序运行过程中临时“组装”SQL语句；支持动态组装SQL语句和动态参数两种形式

(1)使用SQL语句主变量

SQL语句主变量：程序主变量包含的内容是SQL语句的内容，而不是原来保存数据的输入或输出变量；SQL语句主变量在程序执行期间可以设定不同的SQL语句，然后立即执行

[例9] 创建基本表TEST

EXEC SQL BEGIN DECLARE SECTION;

const char \*stmt = "CREATE TABLE test(a int);"; /\* SQL语句主变量 \*/

EXEC SQL END DECLARE SECTION;

... ...

EXEC SQL EXECUTE IMMEDIATE :stmt; /\* 执行语句 \*/

动态SQL：在嵌入式SQL中,提供动态构造SQL功能，分为直接执行/带动态参数/查询类 3种类型；

a.直接执行的动态SQL

只用于非查询SQL语句的执行，应用程序定义一个字符串主变量来存放要执行SQL语句，SQL语句固定部分由程序直接附给主变量，SQL的可变部分由程序提示用户输入，最后,用exec sql EXECUTE IMMEDIATE语句执行字符串主变量中的SQL语句.

exec sql begin declare section;

char sqlstring[200];

exec sql end declare section;

char cond[150]; /\*添入固定的部分\*/

strcpy (sqlstring,”DELETE FROM S WHERE”) /\*提示用户输入查询条件\*/

printf(“Enter search condition:”);

scanf(“%s”,cond);

strcat (sqlstring,cond);

EXEC SQL EXECUTE IMMEDIATE:sqlstring;

……

b.动态参数

动态参数：SQL语句中的可变元素，使用参数符号(?)表示该位置的数据在运行时设定

和主变量的区别：动态参数的输入不是编译时完成绑定，而是通过 (prepare)语句准备主变量和执行(execute)时绑定数据或主变量来完成

使用动态参数的步骤：

声明SQL语句主变量；

准备SQL语句(PREPARE)：

EXEC SQL PREPARE <语句名> FROM <SQL语句主变量>;

执行准备好的语句(EXECUTE)：

EXEC SQL EXECUTE <语句名> [INTO <主变量表>][USING<主变量或常量>];

[例10]向TEST中插入元组。

EXEC SQL BEGIN DECLARE SECTION;

const char \*stmt = "INSERT INTO test VALUES(?);"; /\*声明SQL主变量 \*/

EXEC SQL END DECLARE SECTION;

... ...

EXEC SQL PREPARE mystmt FROM :stmt; /\* 准备语句 \*/

... ...

EXEC SQL EXECUTE mystmt USING 100; /\* 执行语句 \*/

EXEC SQL EXECUTE mystmt USING 200; /\* 执行语句 \*/

(二)存储过程

SQL-invokede routines：存储过程；函数

1.PL/SQL的块结构

(1)PL/SQL：SQL的扩展，增加了过程化语句功能，基本结构是块-->块之间可以相互嵌套；每个块完成一个逻辑操作

(2)PL/SQL块的基本结构

a.定义部分

DECLARE //定义的变量、常量等只能在该基本快中使用

变量、常量、游标、异常等 //当基本块执行结束时，定义就不再存在

b.执行部分

BEGIN

SQL语句过程化SQL的流程控制语句

EXECEPTION //遇到的不能继续执行的情况称为异常

异常处理部分 //出现异常后，采取措施来纠正错误或者报告

END;

2.变量常量的定义

(1)PL、SQL中定义变量的语法格式

变量名 数据类型 [[NOT NULL] := 初值表达式] 或

变量名 数据类型 [[NOT NULL] 初值表达式]

(2)常量的定义类似于变量的定义

常量名 数据类型 CONSTANT := 常量表达式

常量必须要赋值，并在该值存在期间或敞亮的作用域内不能改变

若试图修改常量，PL.SQL将返回一个常量

(3)赋值语句

变量名称 := 表达式

3.控制结构

(1)条件控制语句

a.IF-THEN

IF condition THEN

Sequence\_of\_statements

END IF

b.IF-THEN-ELSE

IF condition THEN

Sequence\_of\_statements1;

ELSE

Sequence\_of\_statements2;

END IF

c.嵌套的IF语句

在THEN和ELSE子句中还可以在包括IF语句

(2)循环控制语句

a.最简单的循环语句LOOP

LOOP

Sequence\_of\_statements

END LOOP

大多数数据库服务器的PL/SQL都提供EXIT、BREAK或LEAVE等循环结束语句，确保LOOP语句块能够结束

b.WHILE-LOOP

WHILE condition LOOP

Sequence\_of\_statements;

END LOOP

每次执行循环体语句之前，要先对条件进行求值，如果条件为真，执行循环体内部的语句序列，反之跳过循环并把控制传递给下一个子句

c.FOR-LOOP

FOR count IN [REVERSE] bound1…bound2 LOOP

Sequence\_of\_statements;

END LOOP

(3)错误处理

如果PL/SQL在执行时出现异常，则应该让程序在产生异常的语句处停下来，根据异常的类型去执行异常处理语句

SQL标准对数据库服务器提供什么样的异常处理做出了建议，要求PL/SQL管理器提供完善的异常处理机制

4.存储过程

(0)PL/SQL块类型：

a.命名块

编译后保存在数据库中，可以被反复使用，运行速度较快。存储过程和函数是命名块

b.匿名块

每次执行时都要进行编译，它不能被存储到数据库中，也不能在其他的PL/SQL块中调用

(1)存储过程的优点

存储过程：有PL/SQL语句书写的过程，经编译和优化后存储在数据库服务器中，使用时只要调用即可

优点：运行效率高；降低了客户机和服务器之间的通信量；方便试试企业规则

(2)存储过程的用户接口

a.创建存储过程

CREATE Procedure 过程名([参数1，参数2，……]) AS

<PL/SQL块>;

过程名：数据库服务器合法的对象标识

参数列表：用名字来表示调用时给出的参数值，必须指定值的数据类型，参数也可以定义输入参数、输出参数或输入/输出参数。默认为输入参数

过程体：是一个<PL/SQL块>，包括声明部分和可执行语句部分

［例11］ 利用存储过程来实现下面的应用: 从一个账户转指定数额的款项到另一个账户中。

CREATE PROCEDURE TRANSFER(inAccount INT， outAccount INT， amount FLOAT)

AS DECLARE

totalDeposit FLOAT;

BEGIN /\* 检查转出账户的余额 \*/

SELECT total INTO totalDeposit

FROM ACCOUNT WHERE ACCOUNTNUM=outAccount;

IF totalDeposit IS NULL THEN /\* 账户不存在或账户中没有存款 \*/

ROLLBACK;

RETURN;

END IF;

IF totalDeposit < amount THEN /\* 账户账户存款不足 \*/

ROLLBACK;

RETURN;

END IF;

UPDATE account SET total=total-amount

WHERE ACCOUNTNUM=outAccount; /\* 修改转出账户，减去转出额 \*/

UPDATE account SET total=total + amount

WHERE ACCOUNTNUM=inAccount; /\* 修改转入账户，增加转出额 \*/

COMMIT; /\* 提交转账事务 \*/

END;

重命名存储过程：ALTER Procedure 过程1 RENAME TO 过程2

b.执行存储过程

CALL/PERFORM Procedure 过程名([参数1，参数2，……]);

使用CALL或者PERFORM等方式激活存储过程的执行

在PL/SQL中，数据库服务器支持在过程体中调用其他存储过程

［例12］从账户01003815868转一万元到01003813828账户中。

CALL Procedure TRANSFER(01003813828，01003815868，10000)；

c.删除存储过程

DELETE PROCEDURE 过程名();

(3)游标

在PL/SQL中，如果SELECT语句只返回一条记录，可以将该结果存放发哦变量中，当产讯返回多条记录时，就要使用游标对结果集进行处理，一个游标与一个SQL语句相关联，PL/SQL中的游标由PL/SQL引擎管理

(三)ODBC编程(开放式数据库连接Open Data Base Connectivity)

ODBC优点：移植性好，能同时访问不通的数据库，共享多个数据资源

(1)数据库互联概述

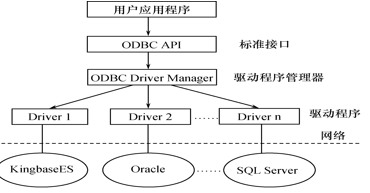
ODBC产生的原因：由于不同的数据库管理系统的存在，在某个RDBMS下辨析的应用程序就不能再另一个RDBMS下运行；许多应用程序需要共享多个部门的数据资源，访问不同的RDBMS

ODBC是微软公司开放服务体系(Windows Open Services Architecture,WOSA)中有关数据库的一个组成部分，提供了一组访问数据库的标准API

ODBC约束力：规范应用开发，规范RDBMS应用接口

(2)ODBC工作原理概述

ODBC应用系统的体系结构：用户应用程序，驱动程序管理器，数据库驱动程序，ODBC数据源管理



a.应用程序

包括：请求连接数据库；向数据源发送SQL语句；为SQL语句执行结果分配内存空间，定义所读取的数据格式；获取数据库操作结果，或处理错误；惊醒数据处理并向用户提交处理结果；请求事务的提交和回滚操作；断开与数据源的连接

b.驱动程序管理器

用来管理各种驱动程序

包含在DOBC32.DLL中；管理应用程序和驱动程序之间的通信；建立、配置或删除数据源并查看系统当前所暗转的数据库ODBC驱动程序；主要功能有装载ODBC驱动程序、选择和连接正确的驱动程序、管理数据源、检查ODBC调用参数的合法性、记录ODBC函数的调用等

c.数据库驱动程序

ODBC通过驱动程序来提供应用系统与数据库平台的独立性

c.1 ODBC应用程序不能直接存取数据库

其各种操作请求有驱动程序管理器提交给某个RDBMS的ODBC驱动程序

通过调用驱动程序所支持的函数来存取数据库

数据库的操作结果也通过驱动程序返回给应用程序

如果应用程序要操纵不同的数据库，就要动态地链接到不同的驱动程序上

c.2 ODBC驱动程序类型

单束：数据源和应用程序在同一台机器上；驱动程序直接完成对数据文件的I/O操作；驱动程序相当于数据管理器

多束：支持客户机/服务器、客户机/应用服务器/数据库服务器等网络环境下的数据访问；由数据驱动完成数据库访问请求提交的结果和结果集接收；应用程序使用驱动程序提供的结果集管理接口操纵执行后的结果数据

d. ODBC数据源管理

数据源是最终用户需要访问的数据，包含了数据库位置和数据库类型等信息，是一种数据连接的抽象

数据源对最终用户是透明的：ODBC给每个被访问的数据源指定唯一的数据源名(Data Source Name,简称DSN)，并映射到所有必要的、用来存取数据的低层软件；在连接中，用数据源名来代表用户名、服务器名、所连接的数据库名等；最终用户无需知道DBMS或其他数据管理软件、网络以及有关ODBC驱动程序的细节

例如，假设某个学校在MS SQL Server和KingbaseES上创建了两个数据库：学校人事数据库和教学科研数据库。

学校的信息系统要从这两个数据库中存取数据；为方便与两个数据库连接，为学校人事数据库创建一个数据源名PERSON，为教学科研数据库创建一个名为EDU的数据源。当要访问每一个数据库时，只要与PERSON和EDU连接即可,不需要记住使用的驱动程序、服务器名称、数据库名

(3)ODBC API基础

a.ODBC应用程序接口的一致性

API一致性：API一致性有核心级、拓展1级、拓展2级

语法一致性：语法一致性级别有最低限度SQL语法级、核心SQL语法级、拓展SQL语法级

b.函数概述

ODBC 3.0 标准提供了76个函数接口：

分配和释放环境句柄、连接句柄、语句句柄；

连接函数(SQLDriverconnect等)；

与信息相关的函数(如获取描述信息函数SQLGetinfo、SQLGetFuction)；

事务处理函数(如SQLEndTran)；

执行相关函数(SQLExecdirect、SQLExecute等)；

编目函数，ODBC 3.0提供了11个编目函数如SQLTables、SQLColumn等，应用程序可以通过对编目函数的调用来获取数据字典的信息如权限、表结构等

ODBC 1.0和ODBC 2.x、ODBC 3.x函数使用上有很多差异

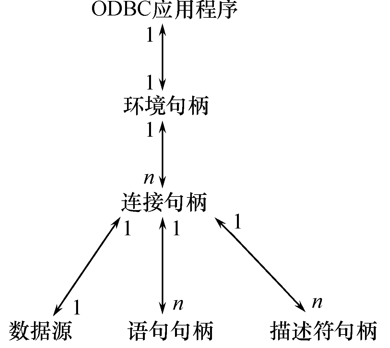
MFC ODBC对较复杂的ODBC API进行了封装，提供了简化的调用接口

c.句柄及其属性

句柄是32位整数值，代表一个指针

ODBC 3.0中句柄分类：环境句柄；连接句柄；语句句柄；描述符句柄；

应用程序句柄之间的关系：



c.1每个ODBC应用程序需要建立一个ODBC环境，分配一个环境句柄，存取数据的全局性背景如环境状态、当前环境状态诊断、当前在环境上分配的连接句柄等

c.2一个环境句柄可以建立多个连接句柄，每一个连接句柄实现与一个数据源之间的连接

c.3在一个连接中可以建立多个语句句柄，它不只是一个SQL语句，还包括SQL语句产生的结果集和相关信息

c.4在ODBC3.0中又提出了描述符句柄的概念，是描述SQL语句中的参数、结果集列的元数据集合

d.数据类型

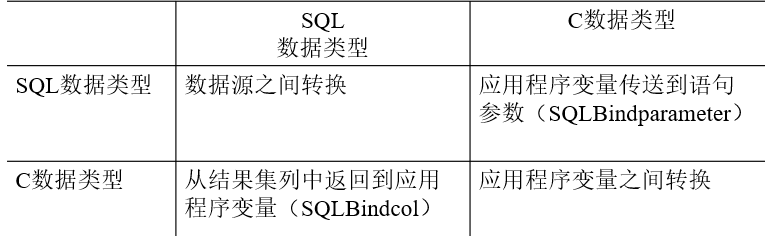
ODBC数据类型

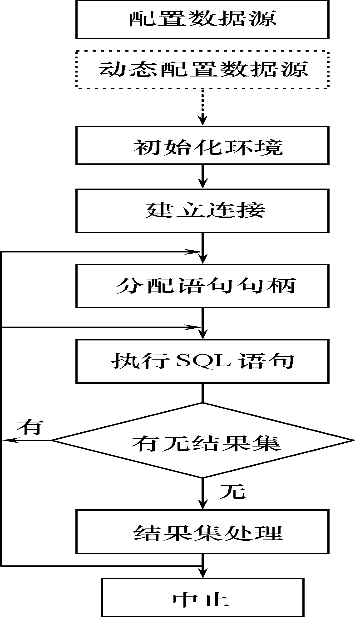
SQL数据类型：用于数据源

C数据类型：用于应用程序的C代码

应用程序可以通过SQLGetTypeInfo来获取不同的驱动程序对于数据类型的支持情况

SQL数据类型和C数据类型之间的转换规则



(4)ODBC的工作流程

引入:

［例13］将KingbaseES数据库中Student表的数据备份到SQL SERVER数据库中。

该应用涉及两个不同的RDBMS中的数据源

使用ODBC来开发应用程序，只要改变应用程序中连接函数(SQLConnect)的参数，就可以连接不同RDBMS的驱动程序，连接两个数据源

在应用程序运行前，已经在KingbaseES和SQL SERVER中分别建立了STUDENT关系表

CREATE TABLE Student

( Sno CHAR(9) PRIMARY KEY，

Sname CHAR(20) UNQUE

Ssex CHAR(2)，

Sage SMALLINT，

Sdept CHAR(20))；

应用程序要执行的操作是：

在KingbaseES上执行SELECT \* FROM STUDENT；

把获取的结果集，通过多次执行

INSERT INTO STUDENT (Sno，Sname，Ssex，Sage，Sddept)

VALUES (?， ?， ?， ?， ?);

插入到SQL SERVER的STUDENT表中

a.配置数据源

配置数据源的两种方法：运行数据源管理工具进行配置；使用Drive Manager提供的CongigDsn函数来增加、修改或删除数据源

在[例13]中，采用了第一种方法创建数据源，因为要同时用到KingbaseES和SQL Server，所以分别建立了两个数据源，将其取名为KingbaseES ODBC 和SQLServer

［例13］创建数据源的详细过程

#include <stdlib.h>

#include <stdio.h>

#include <windows.h>

#include <sql.h>

#include <sqlext.h>

#include <Sqltypes.h>

#define SNO\_LEN 30

#define NAME\_LEN 50

#define DEPART\_LEN 100

#define SSEX\_LEN 5

//创建数据源---第一步：定义句柄和变量

int main(){

/\* Step 1 定义句柄和变量 \*/

//以king开头的表示的是连接KingbaseES的变量

//以server开头的表示的是连接SQLSERVER的变量

SQLHENV kinghenv， serverhenv; //环境句柄

SQLHDBC kinghdbc，serverhdbc; //连接句柄

SQLHSTMT kinghstmt，serverhstmt; //语句句柄

SQLRETURN ret;

SQLCHAR sName［NAME\_LEN］，sDepart［DEPART\_LEN］，

sSex［SSEX\_LEN］， sSno［SNO\_LEN］;

SQLINTEGER sAge;

SQLINTEGER cbAge=0， cbSno=SQL\_NTS， cbSex=SQL\_NTS，

cbName=SQL\_NTS，cbDepart=SQL\_NTS;

b.初始化环境

没有和具体的驱动程序相关联，由Driver Manager来进行控制，并配置环境属性

应用程序通过调用连接函数和某个数据源进行连接后，Driver Manager才调用所连的驱动程序中的SQLAllocHandle，来真正分配环境句柄的数据结构

［例13］创建数据源---第二步：初始化环境

/\* Step 2 初始化环境 \*/

ret=SQLAllocHandle(SQL\_HANDLE\_ENV，SQL\_NULL\_HANDLE， &kinghenv);

ret=SQLAllocHandle(SQL\_HANDLE\_ENV，SQL\_NULL\_HANDLE， &serverhenv);

ret=SQLSetEnvAttr(kinghenv， SQL\_ATTR\_ODBC\_VERSION， (void\*)SQL\_OV\_ODBC3， 0);

ret=SQLSetEnvAttr(serverhenv， SQL\_ATTR\_ODBC\_VERSION， (void\*)SQL\_OV\_ODBC3， 0);

c.建立连接

应用程序调用SQLAllocHandle分配连接句柄，通过SQLConnect、SQLDriverConnect或SQLBrowseConnect与数据源连接

SQLConnect连接函数，输入参数为：配置好的数据源名称；用户ID；口令

[例13]中KingbaseES ODBC为数据源名字，SYSTEM为用户名，MANAGER为用户密码

［例13］创建数据源---第三步：建立连接

/\* Step 3 :建立连接 \*/

ret=SQLAllocHandle(SQL\_HANDLE\_DBC， kinghenv， &kinghdbc);

ret=SQLAllocHandle(SQL\_HANDLE\_DBC， serverhenv， &serverhdbc);

ret=SQLConnect(kinghdbc，"KingbaseES ODBC"，SQL\_NTS， "SYSTEM"，SQL\_NTS， "MANAGER"，SQL\_NTS);

if (!SQL\_SUCCEEDED(ret))//连接失败时返回错误值

return-1;

ret=SQLConnect(serverhdbc， "SQLServer"， SQL\_NTS， "sa"，

SQL\_NTS，"sa"，SQL\_NTS);

if (!SQL\_SUCCEEDED(ret) ) //连接失败时返回错误值

return -1;

d.分配语句句柄

处理任何SQL语句之前，应用程序还需要首先分配一个语句句柄

语句句柄含有具体的SQL语句以及输出的结果集等信息

[例13]中分配了两个语句句柄：一个用来从KingbaseES中读取数据产生结果集(kinghstmt)；一个用来向SQLSERVER插入数据(serverhstmt)

应用程序还可以通过SQLtStmtAttr来设置语句属性(也可以使用默认值)

[例13]中结果集绑定的方式为按列绑定

［例13］创建数据源---第四步

/\* Step 4 :初始化语句句柄 \*/

ret=SQLAllocHandle(SQL\_HANDLE\_STMT， kinghdbc， &kinghstmt);

ret=SQLSetStmtAttr(kinghstmt，SQL\_ATTR\_ROW\_BIND\_TYPE， (SQLPOINTER)

SQL\_BIND\_BY\_COLUMN，SQL\_IS\_INTEGER );

ret=SQLAllocHandle(SQL\_HANDLE\_STMT， serverhdbc， &serverhstmt);

e.执行SQL语句

应用程序处理SQL语句的两种方式：预处理(SQLPrepare、SQLExecute适用于语句的多次执行)；直接执行(SQLExecdirect)

如果SQL语句含有参数，应用程序为每个参数调用SQLBindParameter，并把它们绑定至应用程序变量

应用程序可以直接通过改变应用程序缓冲区的内容从而在程序中动态的改变SQL语句的具体执行

应用程序根据语句的类型进行的处理：有结果集的语句(select或是编目函数)，则进行结果集处理；没有结果集的函数，可以直接利用本语句句柄继续执行新的语句或是获取行计数(本次执行所影响的行数)之后继续执行。

在[例13]中，使用SQLExecdirect获取KingbaseES中的结果集，并将结果集根据各列不同的数据类型绑定到用户程序缓冲区

在插入数据时，采用了预编译的方式，首先通过SQLPrepare来预处理SQL语句，将每一列绑定到用户缓冲区

应用程序可以直接修改结果集缓冲区的内容

［例13］创建数据源---第五步：执行SQL语句

/\* Step 5 :两种方式执行语句 \*/

/\* 预编译带有参数的语句 \*/

ret=SQLPrepare(serverhstmt，"INSERT INTO STUDENT(SNO，SNAME，SSEX， SAGE，SDEPT) VALUES (?， ?， ?， ?， ?)"， SQL\_NTS);

if (ret==SQL\_SUCCESS || ret==SQL\_SUCCESS\_WITH\_INFO){

ret=SQLBindParameter(serverhstmt，1，SQL\_PARAM\_INPUT，SQL\_C\_CHAR，SQL\_CHAR， SNO\_LEN， 0， sSno， 0， &cbSno);

ret=SQLBindParameter(serverhstmt，2，SQL\_PARAM\_INPUT， SQL\_C\_CHAR，SQL\_CHAR， NAME\_LEN， 0， sName， 0， &cbName);

ret=SQLBindParameter(serverhstmt，3，SQL\_PARAM\_INPUT，SQL\_C\_CHAR，SQL\_CHAR， 2， 0， sSex， 0， &cbSex);

ret=SQLBindParameter(serverhstmt，4，SQL\_PARAM\_INPUT，

SQL\_C\_LONG，SQL\_INTEGER， 0， 0， &sAge， 0， &cbAge);

ret=SQLBindParameter(serverhstmt，5，SQL\_PARAM\_INPUT，SQL\_C\_CHAR，

SQL\_CHAR， DEPART\_LEN， 0， sDepart， 0， &cbDepart);

}

/\*执行SQL语句\*/

ret=SQLExecDirect(kinghstmt，"SELECT \* FROM STUDENT"，SQL\_NTS);

if (ret==SQL\_SUCCESS || ret==SQL\_SUCCESS\_WITH\_INFO)

{

ret=SQLBindCol(kinghstmt， 1， SQL\_C\_CHAR，sSno，SNO\_LEN，&cbSno);

ret=SQLBindCol(kinghstmt， 2， SQL\_C\_CHAR，sName，NAME\_LEN， &cbName);

ret=SQLBindCol(kinghstmt， 3， SQL\_C\_CHAR， sSex， SSEX\_LEN， &cbSex);

ret=SQLBindCol(kinghstmt， 4， SQL\_C\_LONG， &sAge， 0， &cbAge);

ret=SQLBindCol(kinghstmt， 5， SQL\_C\_CHAR， sDepart， DEPART\_LEN， &cbDepart);

}

f.结果集处理

应用程序可以通过SQLNumResultCols来获取结果集中的列数

通过SQLDescribeCol或是SQLColAttrbute函数来获取结果集每一列的名称、数据类型、精度和范围

ODBC中使用游标来处理结果集数据

ODBC中游标类型：

forward-only游标，是ODBC的默认游标类型

可滚动(scroll)游标：静态(static)；动态(dynamic)；码集驱动(keyset-driven)；混合型(mixed)

结果集处理步骤：

ODBC游标打开方式不同于嵌入式SQL，不是显式声明而是系统自动产生一个游标(Cursor)，当结果集刚刚生成时，游标指向第一行数据之前

应用程序通过SQLBindCol，把查询结果绑定到应用程序缓冲区中，通过SQLFetch或是SQLFetchScroll来移动游标获取结果集中的每一行数据

对于如图像这类特别的数据类型当一个缓冲区不足以容纳所有的数据时，可以通过SQLGetdata分多次获取

最后通过SQLClosecursor来关闭游标

［例13］创建数据源---第六步：结果集处理

/\* Step 6 ：处理结果集并执行预编译后的语句\*/

while ( (ret=SQLFetch(kinghstmt) ) !=SQL\_NO\_DATA\_FOUND) {

if(ret==SQL\_ERROR) printf("Fetch error＼n");

else

ret=SQLExecute(serverhstmt);

}

g.中止处理

应用程序中止步骤：首先释放语句句柄；释放数据库连接；与数据库服务器断开；释放ODBC环境

［例13］创建数据源---第七步：中止处理

/\* Step 7 中止处理\*/

SQLFreeHandle(SQL\_HANDLE\_STMT，kinghstmt);

SQLDisconnect(kinghdbc);

SQLFreeHandle(SQL\_HANDLE\_DBC， kinghdbc);

SQLFreeHandle(SQL\_HANDLE\_ENV，kinghenv);

SQLFreeHandle(SQL\_HANDLE\_STMT， serverhstmt);

SQLDisconnect(serverhdbc);

SQLFreeHandle(SQL\_HANDLE\_DBC， serverhdbc);

SQLFreeHandle(SQL\_HANDLE\_ENV， serverhenv);

return 0;

}

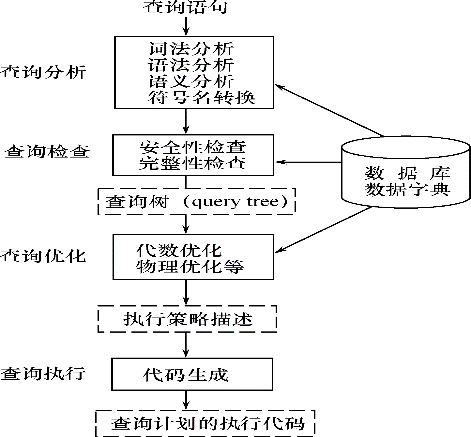
**九、关系查询处理和查询优化**

目的：RDBMS的查询处理步骤；查询优化的概念；基本方法和技术

查询优化分类：代数优化；物理优化

(一)关系数据库系统的查询处理

1.查询处理步骤



(1)查询分析

对查询语句进行扫描、词法分析、语法分析

从查询语句中识别出语言符号

进行语法检查和语法分析

(2)查询检查

根据数据字典对合法的查询语句进行予以检查

根据数据字典中的用户权限和完整性约束定义对用户的存取权限进行检查

检查通过后把SQL查询语句转换成等价的关系代数表达式

RDBMS一般都用查询书(语法分析树)来表示拓展的关系代数表达式

把数据库对象的外部名称转换为内部表示

(3)查询优化

查询优化：选择一个高效执行的查询处理策略

查询优化分类：代数优化-->关系代数表达式的优化

物理优化-->存取路径和底层操作算法的选择

查询优化方法选择的依据：基于个则；基于代价；基于语义

(4)查询执行

依据优化器得到的执行策略生成查询计划

代码生成器生成执行查询计划的代码

2.实现查询操作的算法示例

(1)选择操作的实现

［例1］Select \* from student where <条件表达式> ；

考虑<条件表达式>的几种情况：

C1：无条件；

C2：Sno＝'200215121'；

C3：Sage>20；

C4：Sdept＝'CS' AND Sage>20；

典型实现方法：

a.简单的全表扫描的方法

对查询的基本表顺序扫描，逐一检查每个元组是否满足选择条件，把满足条件的元组作为结果输出

适合小表，不适合大表

b.索引(或散列)扫描方法

适合选择条件的属性上有索引(例如B+树索引或者Hash索引)

通过u索引先找到满足条件的元组主码或元组指针，在通过元组指针直接在查询的基本表中找到元组

［例1-C2］以C2为例，Sno＝‘200215121’，并且Sno上有索引(或Sno是散列码)

使用索引(或散列)得到Sno为‘200215121’ 元组的指针

通过元组指针在student表中检索到该学生

［例1-C3］以C3为例，Sage>20，并且Sage 上有B+树索引

使用B+树索引找到Sage＝20的索引项，以此为入口点在B+树的顺序集上得到Sage>20的所有元组指针

通过这些元组指针到student表中检索到所有年龄大于20的学生。

［例1-C4］以C4为例，Sdept＝‘CS’AND Sage>20，如果Sdept和Sage上都有索引：

算法一：分别用上面两种方法分别找Sdept＝‘CS’的一组元组指针和Sage>20的另一组元组指针

求这2组指针的交集

到student表中检索

得到计算机系年龄大于20的学生

算法二：找到Sdept＝‘CS’的一组元组指针，

通过这些元组指针到student表中检索

对得到的元组检查另一些选择条件(如Sage>20)是否满足

把满足条件的元组作为结果输出。

(2)连接操作的实现

连接操作时查询处理中最耗时的操作之一

此处只讨论等值连接(或自然连接最常用的实现算法)

［例2］SELECT \* FROM Student，SC WHERE Student.Sno=SC.Sno；

a.嵌套循环方法(nested loop)

对外层循环(Student)的每一个组(s)，检索内存循环(SC)中的每一个元组(SC)

检查这两个元组在连接属性(sno)上是否相等

如果满足连接条件，则串接后作为结果输出，知道外层循环中额度元组处理完为止

b.排序-合并方法(sort-merge join/merge join)

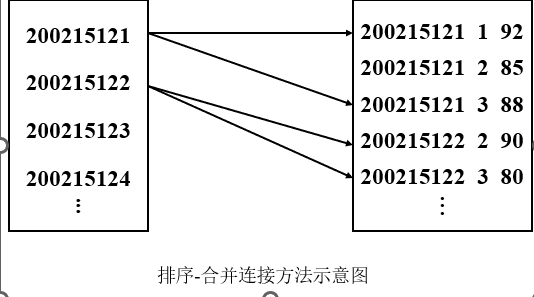
适合连接的诸表已经排好序的情况

排序-合并连接方法的步骤：如果连接的表没有排好序，先对Student表和SC表按连接属性Sno排序

取Student表中第一个Sno，一次扫描SC表中具有相同Sno的元组

当扫描到Sno不相同的第一个SC元组时，返回Student表扫描它的下一个元组，在扫描SC表中具有相同Sno的元组，把它们连接起来

重复上述步骤知道Student表扫描完



Student和SC表都只扫描一次

如果两个表原来无序，执行时间要加上对这两个表的排序时间

对于两个大表，先排序后使用排序-合并方法执行连接，总的时间一般仍会大大减少

c.索引连接(index join)方法

步骤：在SC表上建立属性Sno的索引，如果原来没有该索引，对Student中的每一个元组，有Sno值通过SC的索引查找相应的SC元组，把这些SC元组和Student元组连接起来。循环执行上述步骤，知道Student中的元组处理完为止

(4)Hash Join方法

把连接属性作为Hash码，用同一个Hash函数把R和S中的元组散列到同一个Hash文件中

步骤：

划分阶段：对包含较少元组的表进行一遍处理，把它的元组按Hash函数分散到Hash表的桶中

试探阶段：也称为连接阶段：对另一个表(S)进行一遍处理，把S的元组散列到适当的Hash桶中，把元组与桶中所有来自R并与之相匹配的元组连接起来

上述Hash算法前提：假设两个表中较小的表在第一阶段后可以完全放入内存的Hash桶中

以上的算法思想可以推广到更加一般的多个表的连接算法上

(二)关系数据库系统的查询优化

1.概述

(1)查询优化的优点不仅在于用户不必考虑如何最好地表达查询以获得较好的效率，而且在于系统可以比用户程序的“优化”做的更好

a.优化器可以从数据字典中获取许多统计信息，而用户程序则难以获得这些信息

b.如果数据库的物理统计信息改变了，系统可以自动对查询重新优化以选择相适应的执行计划。在非关系系统中必须重写程序，而重写程序在实际应用中往往是不太可能的

c.优化器可以考虑数百中不同的执行计划，程序员一般只能考虑有限的几种可能性

d.优化器包括了很多复杂的优化技术，这些优化技术往往只有最好的程序员才能掌握。系统的自动优化相当于使得所有人都拥有了这些优化技术

(2)RDBMS通过某种代价模型计算出各种查询执行策略的执行代价，之后选取代价最小的执行方案

a.集中式数据库

执行开销主要包括：磁盘存取块数(I/0代价)；处理机时间(CPU代价)；查询的内存开销。其中I/O代价是最主要的

b.分布式数据库

总代价=I/O代价 + CPU代价 + 内存代价 + 通信代价

(3)查询优化的总目标

选择有效的策略；球的给定关系表达式的值；使得查询代价最小(较小)

2.一个实例

[例3] 求选修了2号课程的学生姓名。用SQL表达：

SELECT Student.Sname

FROM Student，SC

WHERE Student.Sno=SC.Sno AND SC.Cno=‘2’；

假定学生-课程数据库中有1000个学生记录，10000个选课记录，其中选修2号课程的选课记录为50个

系统可以用多种等价的关系代数表达式来完成这一查询

(1)Q1=πSname(σStudent.Sno=SC.Sno∧Sc.Cno='2' (Student×SC))

a.计算广义笛卡尔积

把Student和SC的每一个元组连接起来的做法：

把内存中尽可能多的装入某个表(如Student表)的若干块，留出一块存放到另一个表(如SC表)的元组

把SC中的每一个元组和Student中的每一个元组连接，连接后的元组装满一块后就写到中间文件上

从SC中读入一块和内存中的Student元组连接，直到SC表处理完，再读入若干块Student元组，读入一块元组

重复上述过程，直到把Student表处理完

设一个块能装10个Student元组或100个SC元组，在内存块中存放5块Student和1块SC元组，则读取总块数为：

1000/10 + 1000/(10\*5) \* 10000/100 = 100 + 20\*100=2100

其中读Student表100块，读SC表20遍(每5块读取一遍，共一百块)，每遍100块，每秒读写20块，则需要105秒。连接后的元组数为103\*104=107，设每块能装10个元组，则写出这些块要用106/20=5\*104s

b.作选择操作

一次读入连接后的元组，按照选择条件选取满足要求的记录

假定内存处理时间忽略，读取中间文件花费的时间(同写中间文件一样)需要5\*104s

满足条件的元组假设仅50个，均可放在内存

c.做投影操作

把b中的结果在Sname上作投影输出，得到最终结果

第一种情况下执行查询的总时间≈105+2\*5\*104≈105s

所有内存处理时间均忽略不计

(2)Q2=πSname(σSc.Cno='2' (Student  SC))

a.计算自然连接

执行自然连接，读取Student和SC表的策略不变，总的读取块数认为2100块，话费105s

自然连接的结果比第一种大大减少，为104个(算上外连接，即是选课记录的个数),写出这些元组时间为104/10/20=50s，为第一种情况的千分之一

b.读取中间文件块，执行选择运算，时间也为50s

c.把b中结果投影输出

第二种情况总的执行时间≈105+2\*50≈205s

(3)Q3=πSname(Student  σSc.Cno='2'(SC))

a.先对SC表作选择运算，只需要读一遍SC表，存取100块，花费时间为5s，因为满足条件的元组仅50ge，不必使用中间文件

b.读取Student表，把读入的Student元组和内存中的SC元组作连接，也只需读一遍Student表共100块，花费时间为5s

c.把连接结果投影输出

第三种情况总的执行时间≈5+5≈10s

假如SC表的Cno字段上有索引，第一步就不必读取SC元组而只需要读取Cno=‘2’的那些元组(50个)，存取的索引块和SC中妈祖条件的数据块大约总共3~4块

若Student表在Sno上也有索引，第二部也不必读取所有的Student元组，因为满足条件的SC记录仅有50个，涉及最多50个Student记录，读取Student表的块数也可以大大减少

总的存取时间也会进一步减少数秒

把代数表达式Q1变换为Q2、Q3，即有选择和连接操作时，先做选择操作，这样参加连接的元组就可以大大减少，这是代数优化

在Q3中，SC表的选择操作算法有全表扫描和索引扫描2中方法，经过初步计算，索引扫描方法较优，对于Student和SC表的连接，利用Student表上的左营，采用index join代价也较小，这就是物理优化

(三)代数优化

1.关系代数表达式等价变换规则

代数优化策略：通过对关系代数表达式的等价变换来提高查询效率

关系代数表达式的等价：指用相同的关系代替两个表达式中相应的关系所得到的结果是相同的。两个关系表达式E1和E2是等价的，可以即为E1≡E2

常用的等价变换规则：E为关系代数表达式，F是连接运算的条件

(1)连接、笛卡尔积交换律

E1 × E2≡E2 × E1

E1  E2≡E2  E1

E1  E2≡E2  E1

(2)连接、笛卡尔积的结合律

(*E*1 × *E*2) × *E*3≡*E*1 × (*E*2 × *E*3)

(*E*1   *E*2)  *E*3≡*E*1   (*E*2   *E*3)

(3)投影的串接定律

πA1,A2,…An(πB1,B2…Bn(E)) ≡ πA1,A2,…An(E)

Ai，Bi是属性名而且{A1,A2,...,An}构成{B1,B2,...,Bn}的子集

(4)选择的串接定律

σF1 (σF2 (E)) ≡ σF1∧F2 (E)

选择的串接定律说明选择条件可以合并，这样一次就可以检查全部条件

(5)选择与投影操作的交换律

σF(πA1,A2,…An(E))≡ πA1,A2,…An(σF(E))

选择条件F只涉及属性A1，…，An， 若F中有不属于A1，…，An的属性B1，…，Bm则有更一般的规则：

πA1,A2,…An (σF(E))≡πA1,A2,…An(σF(πA1,A2,…An,B1,B2,…Bm(E)))

(6)选择与笛卡尔积的交换律

如果F中涉及的属性都是E1中的属性，则

σF(E1×E2)≡σF(E1)×E2

如果F=F1∧F2，并且F1只涉及E1中的属性，F2只涉及E2中的属性，则由上面的等价变换规则1，4，6可推出：

σF(E1×E2)≡σF1(E1)×σF2(E2)

若F1只涉及E1中的属性，F2涉及E1和E2两者的属性，则仍有

σF(E1×E2)≡ σF2(σF1(E1)×E2)

它使部分选择在笛卡尔积前先做。

(7)选择与并的分配律

设E=E1∪E2，E1，E2有相同的属性名，则

σF(E1∪E2)≡σF(E1)∪σF(E2)

(8)选择与差运算的分配律

若E1与E2有相同的属性名，则

σF(E1-E2)≡σF(E1)-σF(E2)

(9)选择对自然连接的分配律

σF(E1  E2)≡σF(E1)  σF(E2)

F只涉及E1与E2的公共属性

(10)投影与笛卡尔积的分配律

设E1和E2是两个关系表达式，A1，…，An是E1的属性，B1，…，Bm是E2的属性，则

πA1,A2,…An,B1,B2,…Bn(E1×E2)≡πA1,A2,…An(E1)× πB1,B2,…Bn(E2)

(11)投影与并的分配律

设E1和E2有相同的属性名，则

πA1,A2,…An(E1∪E2)≡πA1,A2,…An(E1)∪πA1,A2,…An(E2)

2.查询树的启发式优化

(1)典型的启发式规则

a.选择运算因尽可能先做，在优化策略中这是最重要、最基本的一条

b.把投影运算和选择运算同时进行

如果有若干投影和选择寻塞纳，并且它们都对同一个关系操作，则可以在扫描此关系的同时完成所有这些运算一避免重复扫描关系

c.把投影同其前或后的双目运算结合

d.把某些选择同在它前面要执行的笛卡尔积结合起来成为一个连接运算

e.找出公共子表达式

如果这种重复出现的子表达式的结果不是很大的关系并且从外存中读入这个关系比计算该子表达式的时间少得多，则先计算一次公共子表达式并把结果写入中间文件就是合算的

当查询的是视图时，定义视图的表达式就是公共子表达式的情况

(2)遵循上述启发式规则，应用上面的等价变换公式来优化关系表达式的算法

算法：关系表达式的优化 输入：一个关系表达式的查询树 输出：优化的查询树

方法：

a.利用等价变换规则4把形如σF1∧F2∧。。。∧Fn(E)变换为σF1(σF2(...(σFn(E))...))

b.对每一个选择，利用等价变换规则4~9尽可能把它移到树的叶端

c.对每一个投影利用等价变换规则3,5,10,11中的一般形式尽可能把它移向树的叶端

注：等价变换规则3使部分投影消失，规则5把一个投影分裂成两个，其中一个有可能被移向树的叶端

d.利用等价变换规则3~5把选择和投影的串接合并成单个选择、单个投影或一个选择后跟一个投影。是多个选择或投影能同时执行，或在一次扫描中全部完成

e.把上述得到的语法树的内节点分组，每一个双目运算(×， ，∪，-)和他的所有直接祖先为一组(这些祖先是σ，π运算)

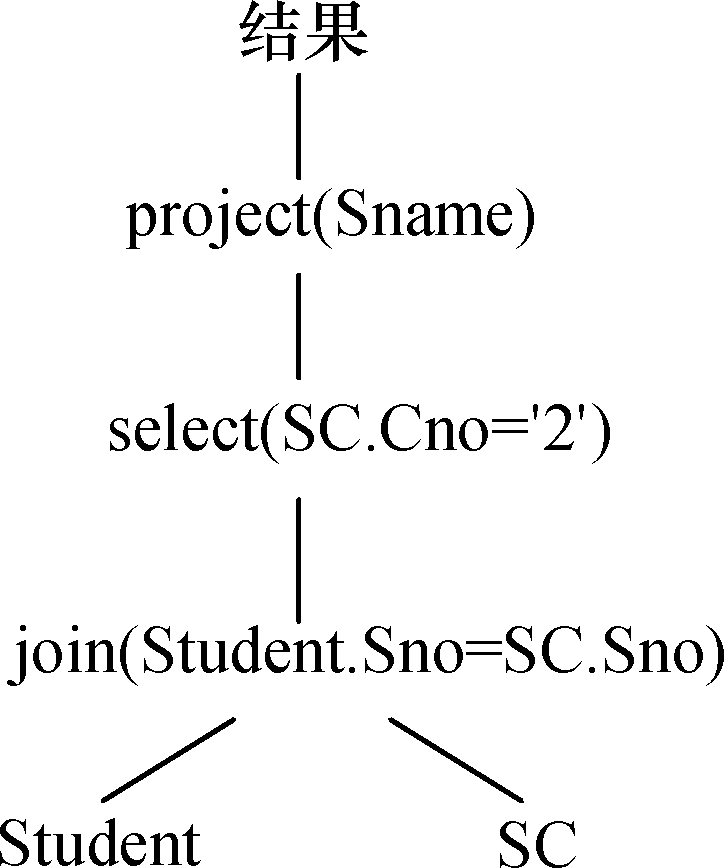
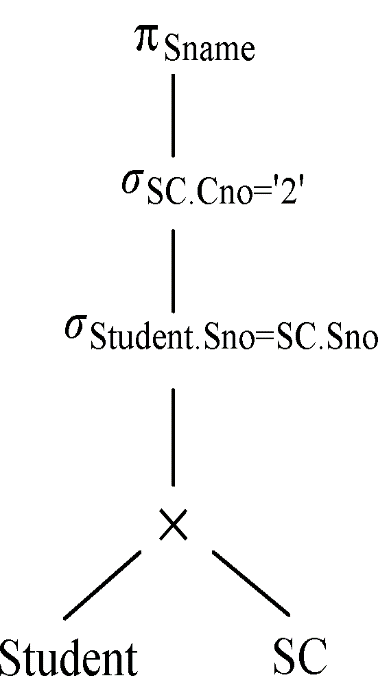
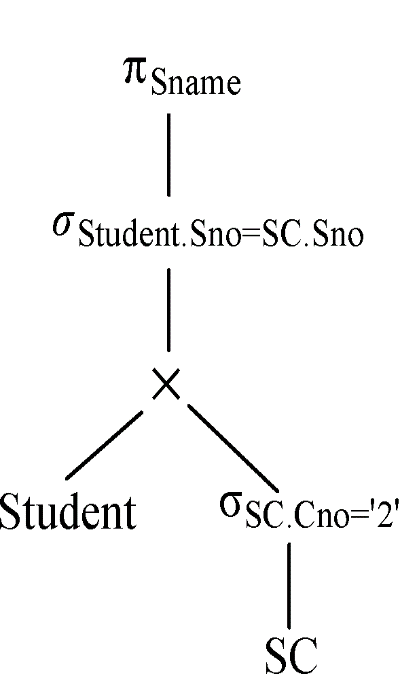
如果其后代直到叶子全是单目运算，则也将他们并入该组，但当双目运算是笛卡尔积(×)，而且后面不是与他组成等值连接的选择时，则不能把选择与这个双目运算组成一组，把这些单目运算单独分为一组

［例4］ 下面给出［例3］中 SQL语句的代数优化示例。

SELECT Student.Sname FROM Student，SC

WHERE Student.Sno = SC.Sno AND SC.Cno=‘2’；

(1) 把SQL语句转换成查询树，如下左图所示

为了使用关系代数表达式的优化法，假设内部表示是关系代数语法树，则上面的查询树如上中图所示。

(2) 对查询树进行优化

利用规则4、6把选择σSC.Cno=‘2’移到叶端，查询树便转换，成上右图所示的优化的查询树。这就是9.2.2节中Q3的查询树表示

(四)物理优化

代数优化改变查询语句中操作的次序和组合，不设计底层的存取路径，对于一个查询语句有许多存取方案，他们的执行效率不同，仅仅进行代数优化时不够的，物理优化就是要选择高效合理的操作算法或存取路径，求得优化的查询计划

选择的方法：基于规则的启发式优化，基于代价估算的优化，两者结合的优化方法

1.基于启发式规则的存取路径选择优化

(1)选择操作的启发式规则

a.对于小关系

使用全表顺序扫描，即使选择列上有索引

b.对于大关系

选择条件是主码=值的查询：查询结果最多是一个元组，可以选择主码索引，一般的RDBMS会自动建立主码索引

选择条件是非主属性=值的查询，并且选择列上有索引，要估算查询结果的元组数目，如果比例较小(<10%)可以使用索引扫描方法，否则还是使用全表顺序扫描。

选择条件是属性上的非等值查询或者范围查询，并且选择列上有索引，要估算查询结果的元组数目，如果比例较小(<10%)可以使用索引扫描方法，否则还是使用全表顺序扫描。

用AND连接的合取选择条件，如果有设计这些属性的组合索引，优先采用组合索引扫描方法。如果某些属性上有一般的索引，则可以用[例1-C4]中介绍的索引扫描方法，否则使用全表顺序扫描

用OR连接的析取选择条件，一般使用全表顺序扫描

(2)连接操作的启发式规则

如果2个表都已经按照连接属性排序， 选用排序-合并方法

如果一个表在连接属性上有索引， 选用索引连接方法

如果上面2个规则都不适用，其中一个表较小，选用Hash join方法

可以选用嵌套循环方法，并选择其中较小的表，确切地讲是占用的块数(b)较少的表，作为外表(外循环的表) 。

理由：设连接表R与S分别占用的块数为Br与Bs，连接操作使用的内存缓冲区块数为K，分配K-1块给外表，如果R为外表，则嵌套循环法存取的块数为Br+( Br/K-1)Bs

显然应该选块数小的表作为外表

2.基于代价的优化

启发式规则优化时定性的选择，适合解释执行的系统：解释执行的系统，优化开销包含在查询总开销之中

编译执行的系统中查询优化和查询执行是分开的，可以采用精细复杂的一些基于代价的优化算法

(1)统计信息

基于代价的优化方法要计算各种操作算法的执行代价，与数据库的状态密切相关

数据字典中存储的优化器需要的统计信息：

对每个基本表：该表的元组总数(N)；元组长度(l)；占用的块数(B)；占用的溢出块数(BO)

对基表的每个列：该列不同值的个数(m)；选择率(f)；如果不同值的分布是均匀的，f＝1/m；如果不同值的分布不均匀，则每个值的选择率＝具有该值的元组数/N；该列最大值

该列最小值；该列上是否已经建立了索引；索引类型(B+树索引、Hash索引、聚集索引)

对索引(如B+树索引)：索引的层数(L)；不同索引值的个数；索引的选择基数S(有S个元组具有某个索引值)；索引的叶结点数(Y)

(2)代价估算实例

全表扫描算法的代价估算公式：

如果基本表大小为B块，全表扫描算法的代价 cost＝B

如果选择条件是码＝值，那么平均搜索代价 cost＝B/2

2. 索引扫描算法的代价估算公式

(1)如果选择条件是码＝值

如［例1-C2］，则采用该表的主索引

若为B+树，层数为L，需要存取B+树中从根结点到叶结点L块，再加上基本表中该元组所在的那一块，所以cost=L+1

(2)如果选择条件涉及非码属性

如［例1-C3］，若为B+树索引，选择条件是相等比较，S是索引的选择基数(有S个元组满足条件)

最坏的情况下，满足条件的元组可能会保存在不同的块上，此时，cost=L+S

如果比较条件是＞，＞＝，＜，＜＝操作，假设有一半的元组满足条件就要存取一半的叶结点，通过索引访问一半的表存储块cost=L+Y/2+B/2

如果可以获得更准确的选择基数，可以进一步修正Y/2与B/2

(c)嵌套循环连接算法的代价估算公式

9.4.1中已经讨论过了嵌套循环连接算法的代价cost＝Br+Bs/(K-1) Br

如果需要把连接结果写回磁盘，cost＝Br+Bs/(K-1) Br +(Frs\*Nr\*Ns)/Mrs

其中Frs为连接选择性(join selectivity)，表示连接结果元组数的比例

Mrs是存放连接结果的块因子，表示每块中可以存放的结果元组数目。

d.排序-合并连接算法的代价估算公式

如果连接表已经按照连接属性排好序，则cost＝Br+Bs+(Frs\*Nr\*Ns)/Mrs。

如果必须对文件排序，需要在代价函数中加上排序的代价，对于包含B个块的文件排序的代价大约是(2\*B)+(2\*B\*log2B)

**十、数据库恢复技术**

(一)事务的基本概念

1.事务定义

定义：一个数据库操作序列，一个不可分割的工作单位，恢复和并发控制的基本单位

事务和程序比较：在关系数据库中，一个事务可以是一条或多条SQL语句，也可以包含一个或多个程序；一个程序通常包含多个事务

(1)显示定义方式：

BEGIN TRANSACTION

SQL 语句1

SQL 语句2

……

COMMIT/ROLLBACK

COMMIT表示提交，即提交事务的所有操作，将事务中所有对数据库的个性协会到磁盘的物理数据库中，实物正常结束。ROLLBACK表示回滚，即在食物运行的过程中发生了某种故障，事务不能继续执行，系统将事务中对数据库的所有已经完成的操作全部撤销，回滚到事务开始时的状态

(2)隐式定义方式

当用户没有显式地定义事务时，DBMS按缺省规定自动划分事务

2.事务的特性

事务的ACID特性：原子性(Atomicity)，一致性(Consistency)，隔离性(Isolation)，持续性(Durability )

(二)数据库恢复概述

故障是不可避免的：

系统故障：计算机软、硬件故障

人为故障：操作员的失误、恶意的破坏等。

数据库的恢复

把数据库从错误状态恢复到某一已知的正确状态(亦称为一致状态或完整状态)

(三)故障的种类

1.事务内部的故障

有的是可以通过事务程序本身发现的(如转账事务)，有的是非预期的

例如，银行转账事务，这个事务把一笔金额从一个账户甲转给另一个账户乙。

BEGIN TRANSACTION

读账户甲的余额BALANCE；

BALANCE=BALANCE-AMOUNT；(AMOUNT 为转账金额)

写回BALANCE；

IF(BALANCE < 0 ) THEN{

打印'金额不足，不能转账'；

ROLLBACK；(撤销刚才的修改，恢复事务)

}

ELSE{

读账户乙的余额BALANCE1；

BALANCE1=BALANCE1+AMOUNT；

写回BALANCE1；

COMMIT；

}

这个例子所包括的两个更新操作要么全部完成要么全部不做。否则就会使数据库处于不一致状态，例如只把账户甲的余额减少了而没有把账户乙的余额增加。

在这段程序中若产生账户甲余额不足的情况，应用程序可以发现并让事务滚回，撤销已作的修改，恢复数据库到正确状态。

事务内部更多的故障是非预期的，是不能由应用程序处理的：运算溢出；并发事务发生死锁而被选中撤销该事务；违反了某些完整性限制等

以后，事务故障仅指这类非预期的故障

事务故障的恢复：撤消事务(UNDO)

2.系统故障

(1)又称为软故障

是指造成系统停止运转的任何时间，使得系统要重新启动：整个系统的正常运行突然被破坏，所有正在运行的事务都非正常终止，不破坏数据库，内存中数据库缓存区的信息全部丢失

(2)常见原因

特定类型的硬件(如CPU故障)，操作系统故障，DBMS代码错误，系统断电

(3)系统故障的恢复

a.发生故障时，事务未提交

强制撤销(UNDO)所有未完成事务

b.发生故障时，事务已经提交，但是缓冲区信息尚未完全回到磁盘上

重做(REDO)所有已经提交的事务

3.介质故障

又称为硬故障，指外存故障：磁盘损坏，磁头碰撞，操作系统的某种潜在错误，瞬时强磁场干扰

恢复：装入数据库发生介质故障前某个时刻的数据副本，重做自此时始的所有成功事务，将这些事务已提交的结果重新计入数据库

4.计算机病毒

一种人为的故障或破坏，是一些恶作剧作者研制的一种计算机程序，可以繁殖和传播

危害：破坏、盗窃系统中的数据，破坏系统文件

(四)恢复的实现技术

恢复操作的基本原理：冗余，利用存储在系统其他地方的冗余数据来重建数据库中已经被破坏或不正确的那部分数据

恢复机制涉及的关键问题：怎样建立数据冗余(数据转储-backup，登录日志文件logging)，怎样利用这些冗余数据来实施数据库恢复

1.数据转储

(1)什么是数据转储

转储是指DBA将整个数据库复制到磁带或者另一个磁盘上保存起来的过程，备用的数据称为后备副本或后援副本

在数据库遭到破坏后可以将后背副本重新装入，重装后备副本只能将数据库恢复到转储时的状态

(2)转储方法

a.静态转储与动态转储

a.1静态转储

在系统中无运行事务时进行的转储操作，转储开始时数据库处于一致性状态，转储期间不允许对数据库的任何存取、修改活动，得到的一定是一个数据一致性的副本

优点：实现简单

缺点：降低了数据库的可用性，转储必须等待正运行的用户事务结束，新的事务必须等转储结束

a.2动态转储

转储操作与用户事务并发进行，转出期间允许对数据库进行存取或修改

优点：不用等待正在运行的用户事务结束，不会影响新事务的运行

缺点：不能保证副本中的数据正确有效

例：在转储期间的某个时刻Tc，系统把数据A=100转储到磁带上，而在下一时刻Td，某一事务将A改为200。转储结束后，后备副本上的A已是过时的数据了

利用动态转储得到的副本进行故障恢复：需要把转储期间各事务对数据库的修改活动登记，建立日志文件，后背副本加上日志文件才能把数据库恢复到某一时刻的正确状态

b.海量转储和增量转储

海量转储：每次转储全部数据库

增量转储：只转储上次转储后更新过的数据

海量转储和增量转储比较：从恢复角度看，使用海量转储得到的后背斧王进行恢复往往更方便，但如果数据库很大，事务处理又很频繁，则增量转储更加实用有效

|  |  |  |
| --- | --- | --- |
| 转储状态  转储方式 | 动态转储 | 静态转储 |
| 海量转储 | 动态海量转储 | 静态海量转储 |
| 增量转储 | 动态增量转储 | 静态增量转储 |

2.登记日志文件

(1)日志文件的格式和内容

日志文件是用来记录事务对数据库的更新操作的文件

日志文件的格式：以记录为单位的日志文件，以数据块为单位的日志文件

a.以记录为单位的日志文件

日志文件中需要登记的内容：各个事务的开始标记(BEGIN TRANSACTION)、结束标记(COMMIT或ROLLBACK)、各个事务的所有更新操作，这些均作为日志文件中的一个日志记录

每条日志记录的内容：事务标识(标明是哪个事务) ，操作类型(插入、删除或修改)，操作对象(记录内部标识)，更新前数据的旧值(对插入操作而言，此项为空值)，更新后数据的新值(对删除操作而言, 此项为空值)

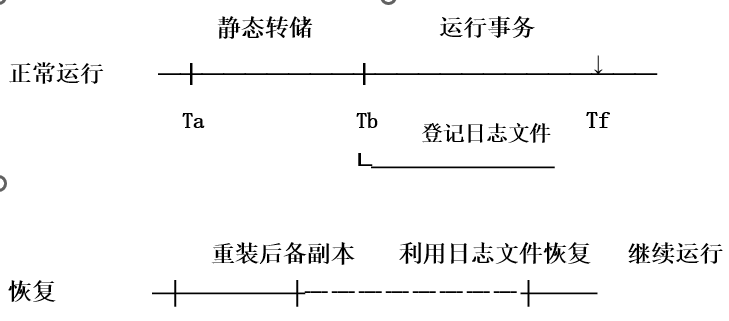
b.以数据块为单位的日志文件

每条日志记录的内容：事务标识(标明是那个事务)，被更新的数据块

(2)日志文件的作用

用来进行事务故障恢复和系统故障恢复，协助后备副本进行介质故障恢复

具体的：事务故障恢复和系统故障恢复必须使用日志文件，在动态转储方式中必须建立日志文件，后备副本和日志文件结合起来才能有效的恢复数据库，在静态转储方式中也可以建立日志文件，当数据库毁坏后可重新转入后援副本把数据库恢复到转储结束时刻的正确状态，之后利用日志文件把已经完成的食物进行重做处理，对故障发生时尚未完成的食物进行撤销处理，这样不必重新运行那些已经完成的事务程序就可以把数据库恢复到故障前某一时刻的正确状态



上图中：系统在Ta时刻停止运行事务，进行数据库转储，在Tb时刻转储完毕，得到Tb时刻的数据库一致性副本，系统运行到Tf时刻发生故障，为恢复数据库，首先由DBA重装数据库后备副本，将数据库恢复至Tb时刻的状态，重新运行自Tb～Tf时刻的所有更新事务，把数据库恢复到故障发生前的一致状态

(3)登记日志文件

基本原则：登记的次序严格按照并行事务执行的时间次序，必须先写日志文件，厚些数据库(写日志文件操作：把表示这个修改的日志记录写到日志文件，写数据库操作：把对数据的修改写到数据库中)

为什么先写日志文件：写数据库和写日志文件是两个不同的操作，在这两个操作之间可能发生故障，如果先写了数据库修改，而在日志文件中没有登记下这个修改，则以后无法恢复这个修改，如果先写日志，但没有修改数据库，按照日志文件恢复时只是多执行一次不必要的UNDO操作，并不会影响数据库的正确性

(五)恢复策略

(1)事务故障的恢复

a.事务故障：

事务在运行至正常终止点前被终止

b.恢复方法：

由恢复子系统利用日志文件撤销(UNDO)此事务对数据库进行的修改

事务故障的恢复由子系统自动完成，对用户是透明的，不需要用户干预

c.恢复步骤：

c.1反向扫描该文件(从最后向前扫描日志文件)，查找该事务的更新操作

c.2对该事务的更新操作执行逆操作，即将日志记录中“更新前的值”写入数据库

插入操作：“更新前的值为空”，则相当于做删除操作

删除操作：“更新后的值为空”，则相当于做插入操作

如果是修改操作，则用修改前的值代替修改后的值

c.3继续反向扫描日志文件，查找该事务的其他更新操作，并做出同样的处理

c.4如此处理直至读到事务的开始标记，事务故障恢复完成

(2)系统故障的恢复

a.系统故障造成数据库不一致状态的原因：

未完成的事务对数据库的更新已写入数据库，已提交事务对数据库的更新还留在缓冲区还没来得及写入数据库

b.恢复方法：

UNDO故障发生时未完成的事务

REDO故障发生时已完成的事务

系统故障的恢复由系统在重新启动时自动完成，不需要用户干预

c.恢复步骤：

c.1正向扫描日志文件(从头扫描日志文件)

重做(REDO)队列：在故障发生前已经提交的事务，这些事务既有BEGIN TRANSACTION记录，也有COMMIT记录

撤销(UNDO)队列：在故障发生时尚未完成的事务，这些事务只有BEGIN TRANSACTION记录，没有相应的COMMIT记录

c.2对撤销(UNDO)队列事务进行撤销(UNDO)处理

反向扫描日志文件，对每个UNDO事务的更新操作执行逆操作， 即将日志记录中“更新前的值”写入数据库

c.3对重做(REDO)队列事务进行重做(REDO)处理

正向扫描日志文件，对每个REDO事务重新执行登记的操作，即将日志记录中“更新后的值”写入数据库

(3)介质故障的恢复

恢复步骤：

a.重装数据库

装入最新的后备数据库副本(离故障发生时刻最近的转储副本)，使数据库恢复到最近一次转储时的一致性状态

对于静态转储的数据库副本，装入数据库即处于一致性状态

对于动态转储的数据库副本，还须同时装入转储时刻的日志文件副本，利用恢复系统故障的方法(UNDO+REDO)，才能将数据库恢复到一致性状态

b.重做已经完成的事务

装入有关的日志文件副本后(转储结束时刻的日志文件副本)，重做已经完成的事务

首先扫描日志文件，找出故障发生事已经提交的事务的标记，将其记入重做队列，之后正向扫描日志文件，对重做队列中的所有事务进行重做处理，即将日志记录中“更新后的值”写入数据库

c.介质故障的恢复需要DBA介入

DBA的工作：重装最近转储的数据库副本和有关的各日志文件副本，执行系统提供的恢复命令

具体的恢复操作任由DBMS完成

(六)具有检查点的恢复技术

(1)问题提出

a.两个问题

搜索整个日志将耗费大量的时间

REDO处理：重新执行，浪费了大量时间

b.解决方案

具有检查点（checkpoint）的恢复技术：在日志文件中增加检查点记录（checkpoint）；增加重新开始文件；恢复子系统在登录日志文件期间动态地维护日志

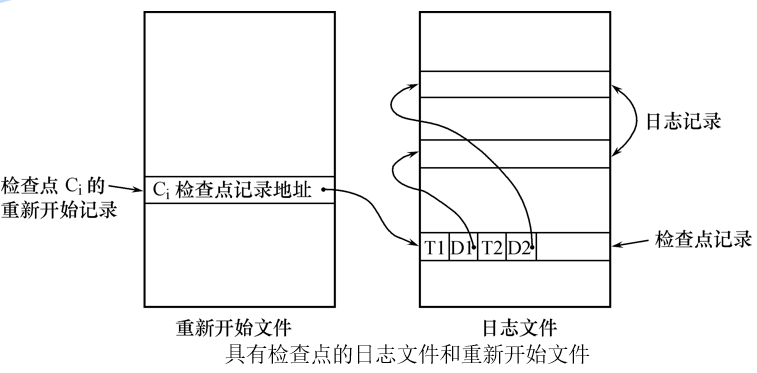
(2)检查点技术

a.检查点记录的内容

建立检查点时刻所有正在执行的事务清单以及这些事务最近一个日志记录的地址

b.重新开始文件的内容

记录各个检查点记录在日志文件中的地址



c.动态维护日志文件的方法

周期性的执行如下操作：建立检查点，保存数据库状态

具体步骤：

c.1将当前日志缓冲区中的所有日志记录写入磁盘的日志文件上

c.2在日志文件中写入一个检查点记录

c.3将当前数据缓冲区的所有数据记录写入磁盘的数据库中

c.4把检查点记录在日志文件中的地址写入一个重新开始文件

恢复子系统可以定期或不定期地建立检查点,保存数据库状态

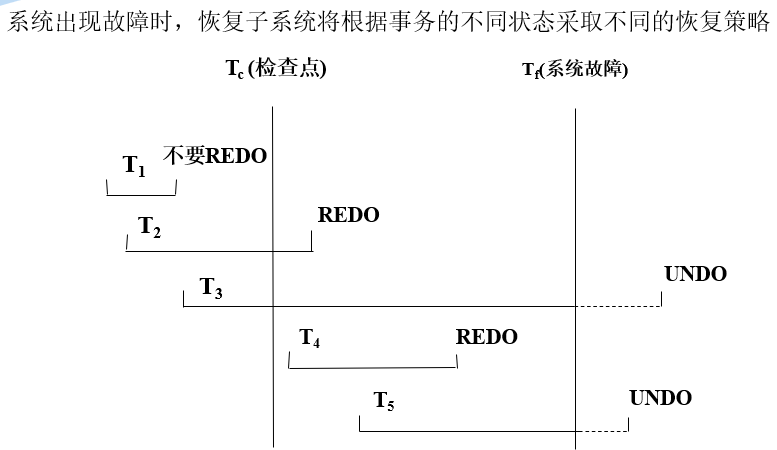
定期；按照预定的一个时间间隔，如每隔一小时建立一个检查点

不定期：按照某种规则，如日志文件已写满一半建立一个检查点

(3)利用检查点的恢复策略

a.使用检查点方法可以改善恢复效率

当事务T在一个检查点之前提交，T对数据库所做的修改已写入数据库，写入时间是在这个检查点建立之前或在这个检查点建立之时，在进行恢复处理时，没有必要对事务T执行REDO操作



状态分析：

T1：在检查点之前提交

T2：在检查点之前开始执行，在检查点之后故障点之前提交

T3：在检查点之前开始执行，在故障点时还未完成

T4：在检查点之后开始执行，在故障点之前提交

T5：在检查点之后开始执行，在故障点时还未完成

恢复策略：

T3和T5在故障发生时还未完成，所以予以撤销

T2和T4在检查点之后才提交，它们对数据库所做的修改在故障发生时可能还在缓冲区中，尚未写入数据库，所以要REDO

T1在检查点之前已提交，所以不必执行REDO操作

b.恢复步骤：

b.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

b.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，REDO-LIST

把ACTIVE-LIST暂时放入UNDO-LIST队列，REDO队列暂为空。

b.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

b.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

(七)数据库镜像

(1)介质故障是对系统影响最为严重的一种故障，严重影响数据库的可用性，原因：

介质故障恢复比较费时

为预防介质故障，DBA必须周期性地转储数据库

(2)提高数据库可用性的解决方案：数据库镜像（Mirror）

DBMS自动把整个数据库或其中的关键数据复制到另一个磁盘上

DBMS自动保证镜像数据与主数据库的一致性

每当主数据库更新时，DBMS自动把更新后的数据复制过去（如下图所示）



(3)出现介质故障时

可由镜像磁盘继续提供使用

同时DBMS自动利用镜像磁盘数据进行数据库的恢复

不需要关闭系统和重装数据库副本(如下图所示)



(4)没有出现故障时

可用于并发操作

一个用户对数据加排他锁修改数据，其他用户可以读镜像数据库上的数据，而不必等待该用户释放锁

(5)频繁地复制数据自然会降低系统运行效率

在实际应用中用户往往只选择对关键数据和日志文件镜像，而不是对整个数据库进行镜像

**十一、并发控制**

(零)问题产生

允许多个用户同时使用的数据库系统，如飞机订票数据库系统，银行数据库系统

特点：在同一时刻并发运行的事务数目可达数百个

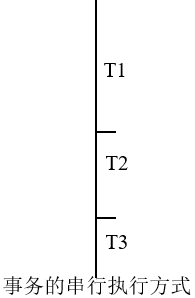
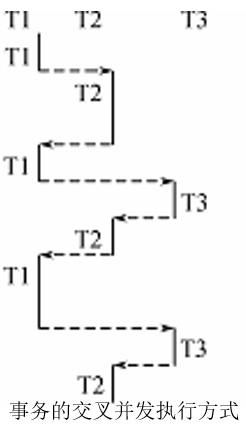
并发控制的基本单位是事务

一、不同的多事务执行方式

1.事务串行执行

每个时刻只有一个事务运行，其他事务必须等到这个事务结束后才能运行

不能充分利用系统资源，发挥数据库共享资源的特点



2.交叉并发方式

在单处理机系统中，系统的并行执行是这些并行事务的并行操作轮流交叉进行，单处理机系统中的并行事务并没有真正的并行运行，但是能够减少处理机的空闲时间，提高系统的效率

3.同时并发方式

多处理机系统中，每个处理机可以运行一个事务，多个处理机可以同时运行多个事务，实现多个事务的真正的并行运行

二、事务并发执行带来的问题

会产生多个事务同时存取统一数据的情况

可能会存取和存储不正确的数据，破坏事务一致性和数据库的一致性

(一)并发控制概述

并发控制的任务：对并发操作进行正确调度，保证事务的隔离性，保证数据库的一致性

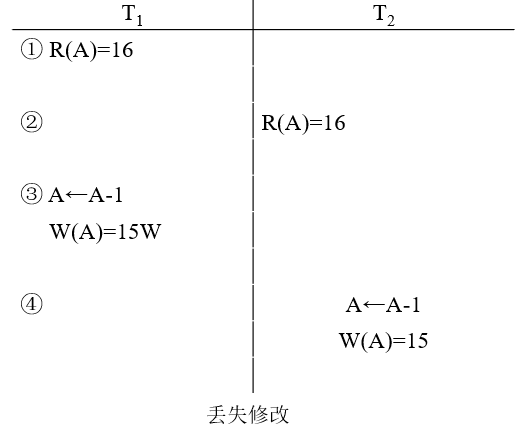
并发操作带来数据的不一致性的实例：

[例1]飞机订票系统中的一个活动序列

1. 甲售票点(甲事务)读出某航班的机票余额A，设A=16；
2. 乙售票点(乙事务)读出同一航班的机票余额A，也为16；
3. 甲售票点卖出一张机票，修改余额A←A-1，所以A为15，把A写回数据库；
4. 乙售票点也卖出一张机票，修改余额A←A-1，所以A为15，把A写回数据库

结果明明卖出两张机票，数据库中机票余额只减少1

这种情况称为数据库的不一致性，是由并发操作引起的，在并发操作情况下，对甲乙两个事务的操作序列的调度是随机的，如果按照上面的调度序列执行，甲事务的修改就丢失，原因是第4步中乙事务修改A并协会后覆盖了甲事务的修改

 并发操作带来的数据不一致性：

丢失修改；不可重复读；读“脏”数据

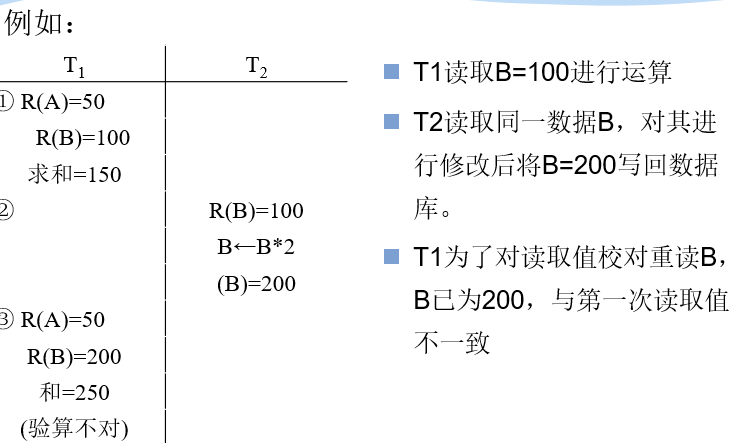
记号：R(x)：读数据x；W(x)：写数据x

(一)丢失修改

两个事务T1、T2读入统一数据并修改，T2的提交结果破坏了T1提交的结果，导致T1的修改被丢失

(二)不可重复读

事务T1读取数据后，事务T2执行更新操作，使T1无法再现前一次读取结果。有三种情况

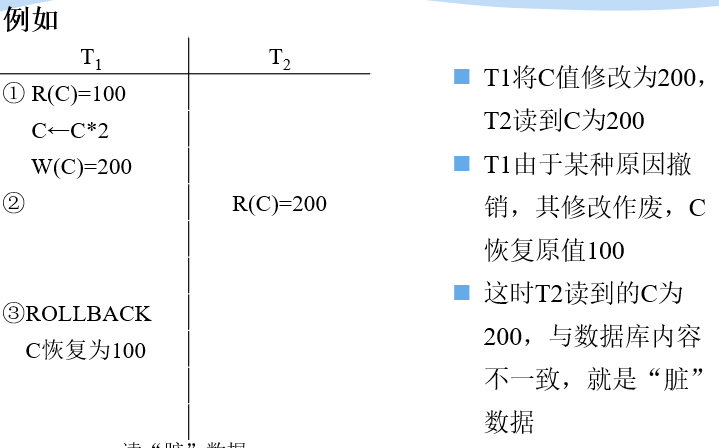
1.事务T1读取某一数据后，事务T2对其做了修改，当事务T1再次读取该数据时,得到有钱一次不同的值

2.事务T1按一定条件从数据库中读取了某些数据记录后，事务T2删除了其中对的部分记录，当T1再次按照相同条件读取数据时，发现某些记录消失了

3. 事务T1按一定条件从数据库中读取了某些数据记录后，事务T2插入了一些记录，当T1再次按照相同的条件读取数据时，发现多了一些记录

后两种不可重复读有时也称为幻影现象

(三)读“脏”数据

 读“脏”数据是指：事务T1修改某一数据后，并将其写回磁盘，事务T2读取统一数据后，T1由于某种原因被撤销，这是T1已修改的数据恢复原值，T2读到的数据就与数据库中的数据不一致，T2读到的数据就是“脏”数据，即不正确的数据

数据不一致性：由于并发操作破坏了事务的隔离性，并发控制就是要用正确的方式调度并发操作，是一个用户事务的执行不受其他事务的干扰，从而造成数据的不一致性

并发控制的主要技术有：封锁，时间戳，乐观控制法，多版本并发控制。常用的DBMS一般都采用封锁方法

(二)封锁

1.封锁的定义

封锁就是事务T在对某个数据对象(例如表、记录等)操作之后，先向系统发出请求，对其枷锁

加锁后事务T就对该数据对象有了一定的控制，在事务T释放它的锁之前，其他的事务不能更新此数据对象

2.基本封锁类型

一个事务对某个数据对象加锁后究竟拥有什么样的控制由封锁的类型决定

基本封锁类型：排他锁：X锁；共享锁：S锁

(1)排他锁(X锁)

排他锁又称为写锁

若事务T对数据对象加上X锁，则只允许T读取和修改A，其他任何事物都不能再对A加任何锁，知道T释放A上的锁

保证奇特事务在T释放A上的锁之前不能再读取和修改A

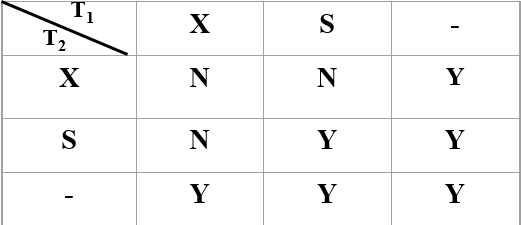
(2)共享锁(S锁)

共享锁又称为读锁

若事务T对数据对象A加上S锁，则事务T可以读A但不能修改A，其他事务只能再对A加S锁，而不能加X锁，知道T释放A上的S锁

保证其他事务可以读A，但是在T释放A上的S锁之前不能对A做任何修改

3.锁的相容矩阵

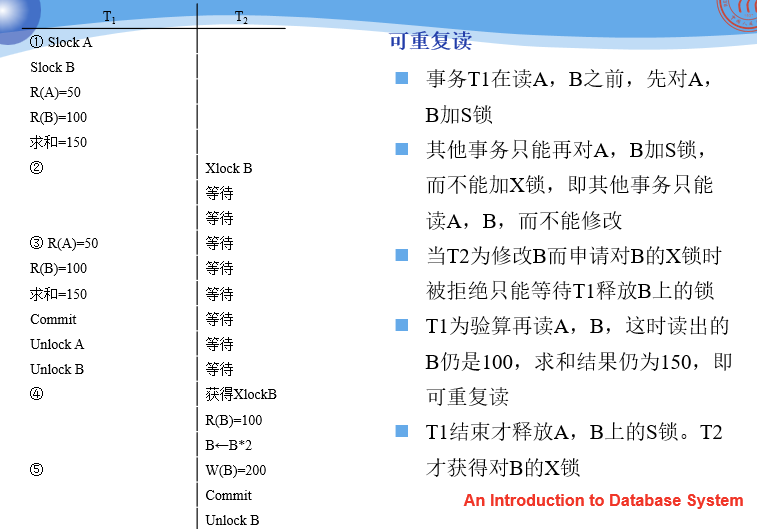


4.使用封锁机制解决三类问题

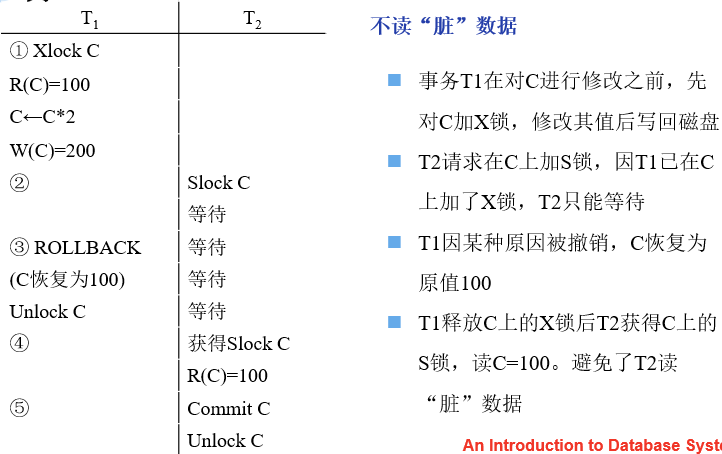
(1)丢失修改



(2)不可重复读



(3)读“脏”数据



5.封锁协议

在使用X锁和S锁对数据对象加锁时，需要约定一些规则，这些规则称为封锁协议

(1)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，知道事务结束才释放

防止丢失修改，但如果仅仅是读数据而不对进行修改，是不需要加锁的，所以不能保证可重复读和不读“脏”数据

(2)二级封锁协议

在一级封锁协议基础上增加事务T在读取数据R之前必须先对其加S锁，读完后立即释放S锁

防止丢失修改，防止读脏数据，不能保证可重复读

(3)三级协议

在一级封锁协议基础上增加事务T在读取数据R之前必须先对其加S锁，直到事务结束才释放

防止丢失修改，防止读脏数据，保证可重复读

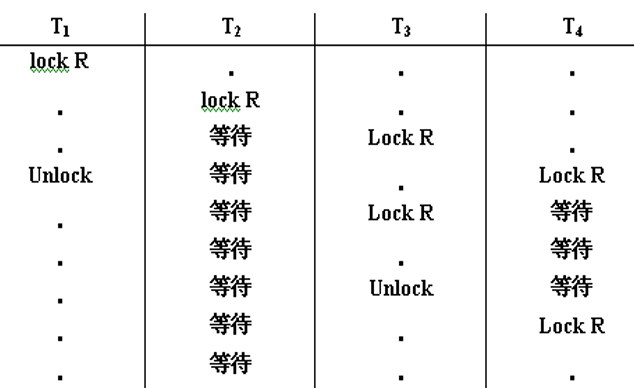
(三)活锁和死锁

1.活锁

(1)定义

事务T1封锁了数据R，事务T2又请求封锁R，于是T2等待，T3也请求封锁R，当T1释放了R上的封锁之后系统首先批准了T3的请求，T2仍然等待，T4又请求封锁R，当T3释放了R上的封锁之后系统又批准了T4的请求……T2有可能永远等待，这就是活锁的情形

多个事务对同一数据请求加锁，可能有一个事务一直加不上的情况



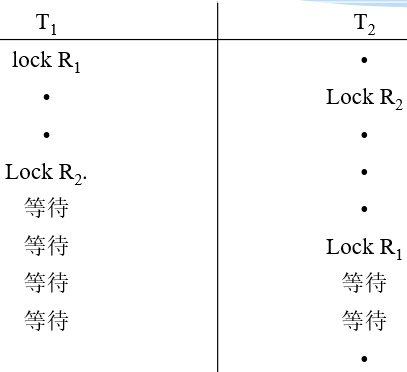
(2)避免活锁：采用先来先服务的策略

当多个事务请求封锁同一数据对象时，按请求封锁的先后次序对这些事务排队，该数据对象上的锁一旦释放，首先批准申请队列中第一个事务获得锁

2.死锁

(1)定义

事务T1封锁了数据R1，T2封锁了数据R2，T1又请求封锁R2，因T2已封锁了R2，于是T1等待T2释放R2上的锁，接着T2又申请封锁R1，因T1已封锁了R1，T2也只能等待T1释放R1上的锁，这样T1在等待T2，而T2又在等待T1，T1和T2两个事务永远不能结束，形成死锁



(2)预防死锁

产生死锁的原因是啊两个或多个事务都已封锁了数据对象，让后又都请求对已为其他事务封锁的数据对象枷锁，总而出现死等待

预防死锁的发生就是要破坏产生死锁的条件

a.一次封锁法

要求每个事务必须一次将所有要使用的数据全部加锁，否则就不能执行

问题：降低系统并发度，难于实现精确确定封锁对象，只能扩大封锁范围，进一步降低系统并发度

b.顺序封锁法

预先对数据对象规定一个封锁顺序，要求所有事务都按照这个顺序实行封锁

问题：数据库系统中封锁的数据对象极多，并且在不断变化，维护这样的资源的封锁顺序较难，成本较高。事务的封锁请求随着事务的执行而动态变化，难以事先确定每一个事物要封锁哪些数据对象

(3)死锁的诊断与解除

在操作系统中广为采用的预防死锁的策略并不很适合数据库的特点，DBMS在解决死锁的问题上更普遍采用的是诊断并解除死锁的方法

a.超时法

如果一个事务的等待时间超过了规定的时限，就认为发生了死锁

优点：实现简单

缺点：时限设置的如果太短，可能误判死锁，时限设置的如果太长，死锁发生后不能及时发现

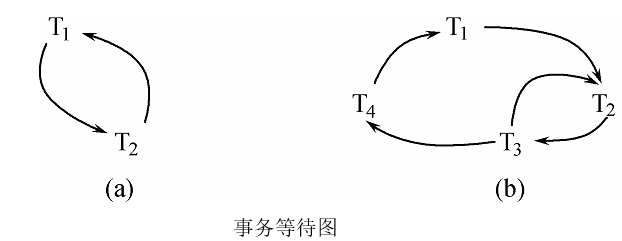
b.事务等待图法

用事务等待图动态反映所有事物的等待情况

事务等待图是一个有向图G=(T,U)，T为结点的集合，每个结点表示正在运行的事务。U为边的集合，每条边表示事物等待的情况。若T1等待T2则T1，T2之间划一条有向边，从T1指向T2

图a中，事务T1等待T2，T2等待T1，产生了死锁

图b中，事务T1等待T2，T2等待T3，T3等待T4，T4又等待T1，产生了死锁。同时T3还可能等待T2，在大回路中还有小回路



并发控制子系统周期性的(如每隔几秒)生成事务等待图，监测事务，如果发现图中存在回路，则表示系统中出现了死锁

c.死锁的解除

选择一个处理死锁代价最小的事务，将其撤销，释放此事务持有的所有的锁，使其它事务能继续运行下去

(四)并发调度的可串行性

DBMS对并发事务不同的调度可能会产生不同的结果

1.可串行化调度

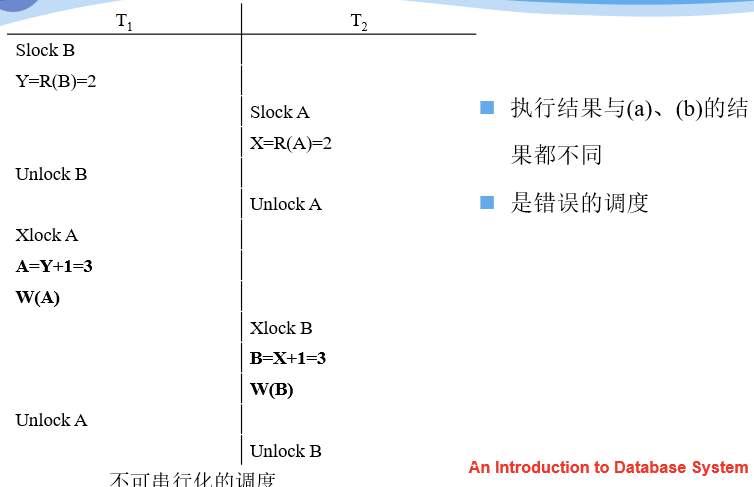
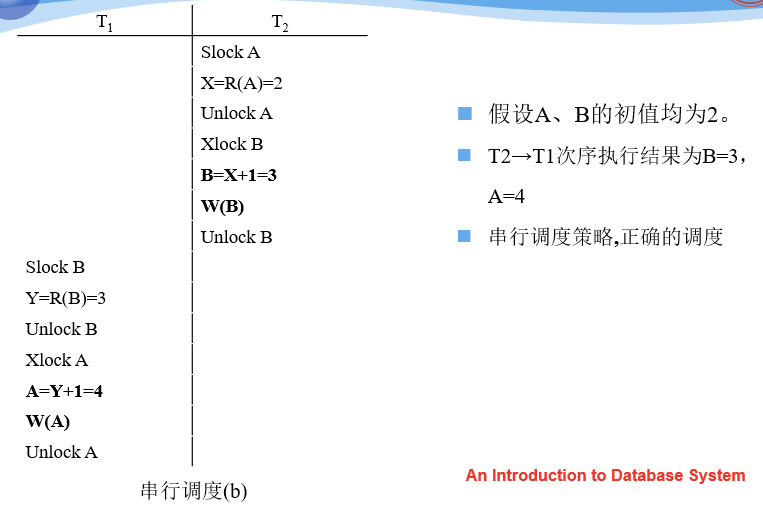
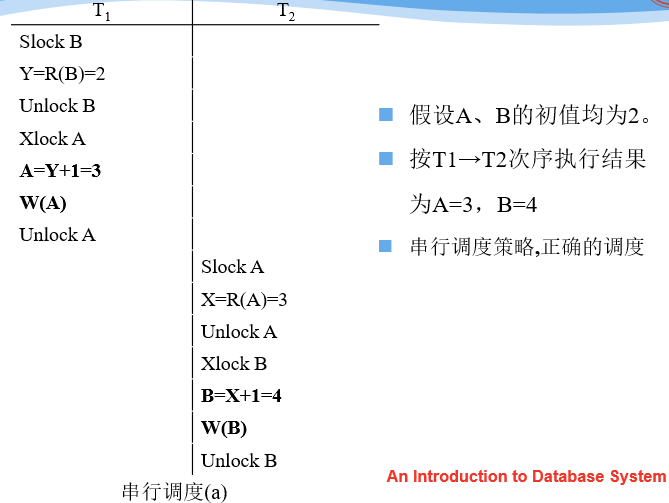
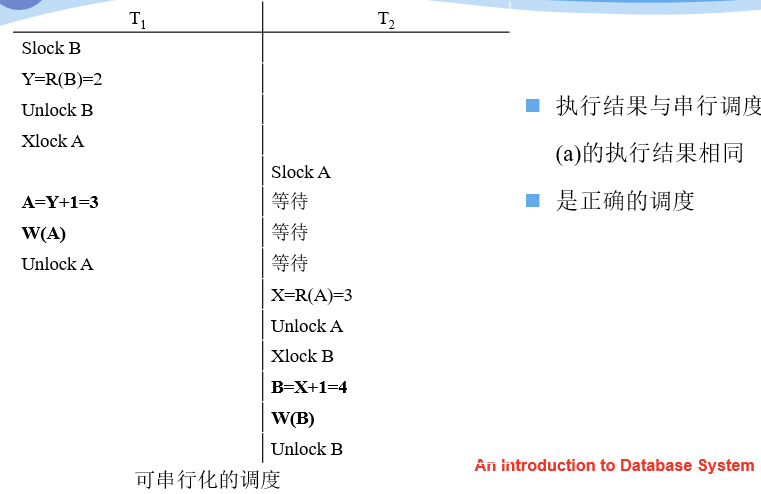
多个事务的并发执行是正确的，当且仅当其结果与按某一次串行执行这些事务时的结果相同

可串行性：是并发事务正确调度的准则，一个给定的并发调度，当且仅当它是可串行话的才认为是正确调度

[例]现在有两个事务，分别包含下列操作：

事务T1：读B；A=B+1；写回A，事务T2：读A；B=A+1；写回B

现给出对这两个事务不同的调度策略

2.冲突可串行化调度

a.定义

一个调度Sc在保证冲突操作的次序不变的情况下，通过交换两个事务不冲突操作的次序得到另一个调度Sc’，如果Sc’是串行的，称调度Sc为可串行化的调度

若一个调度是冲突可串行化，一定是可串行化的调度

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

Ri(x) 和 Wj(x) 事务Ti读x，事务Tj写x

Wi(x) 和 Wj(x) 事务Ti写x，事务Tj写x

其它冲突是不冲突操作，不同事务的冲突操作和同一事务的两个操作不能交换

[例］今有调度Sc1=r1(A)w1(A)r2(A)w2(A)r1(B)w1(B)r2(B)w2(B)

把w2(A)与r1(B)w1(B)交换，得到:r1(A)w1(A)r2(A)r1(B)w1(B)w2(A)r2(B)w2(B)

再把r2(A)与r1(B)w1(B)交换：Sc2＝r1(A)w1(A)r1(B)w1(B)r2(A)w2(A)r2(B)w2(B)

Sc2等价于一个串行调度T1，T2，Sc1冲突可串行化的调度

b.冲突可串行化调度是可串行化调度的充分条件，不是必要条件，还有不满足冲突可串行化条件的可串行化调度

[例]有3个事务：T1=W1(Y)W1(X)，T2=W2(Y)W2(X)，T3=W3(X)

调度L1=W1(Y)W1(X)W2(Y)W2(X)W3(X)是一个串行调度

调度L2=W1(Y)W2(Y)W2(X)W1(X)W3(X)不满足冲突可串行化。

但是调度L2是可串行化的，因为L2执行的结果与调度L1相同，Y的值都等于T2的值，X的值都等于T3的值

(五)两段锁协议

1.封锁协议

运用封锁方法时，对数据对象加锁时需要约定一些规则：何时申请封锁，持锁时间，何时释放封锁等

两段封锁协议(Two-Phase Locking,2PL)：是最常用的一种封锁协议，理论上证明使用两段封锁协议产生的是可串行化调度

2.两段锁协议

指所有事务必须分两个阶段对数据项加锁和解锁，在对任何数据进行读、写操作之前，事务首先要获得对该数据的封锁，在释放一个封锁之后，事务不再申请和获得其它封锁

两段锁的含义：事务分为两个阶段

a.第一个阶段是获得封锁，也称为扩展阶段，事务可以申请获得任何数据项上的任何类型的锁，但是不能释放任何锁

b.第二个阶段是释放封锁，也称为收缩阶段，事务可以释放任何数据项上的任何类型的锁，但是不能再申请任何锁

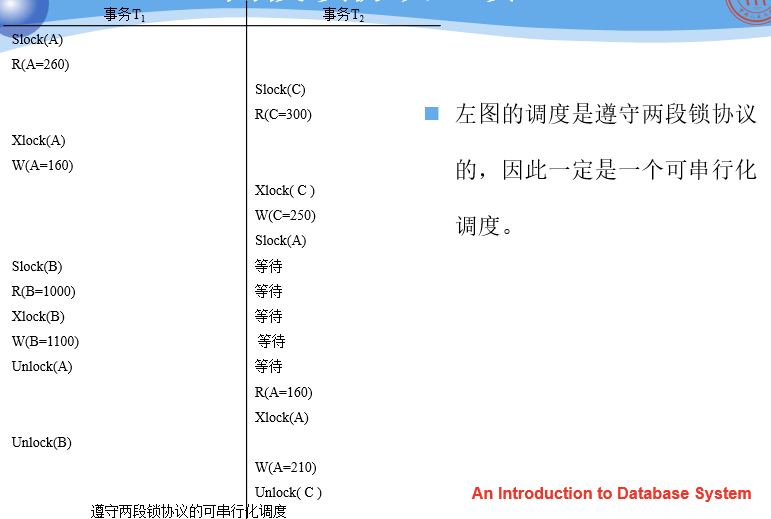
例：事务Ti遵守两段锁协议，其封锁序列是 ：

Slock A Slock B Xlock C Unlock B Unlock A Unlock C；

|← 扩展阶段 →| |← 收缩阶段 →|

事务Tj不遵守两段锁协议，其封锁序列是：

Slock A Unlock A Slock B Xlock C Unlock C Unlock B；



事务遵守两段锁协议是可串行化调度的充分条件，而不是必要条件

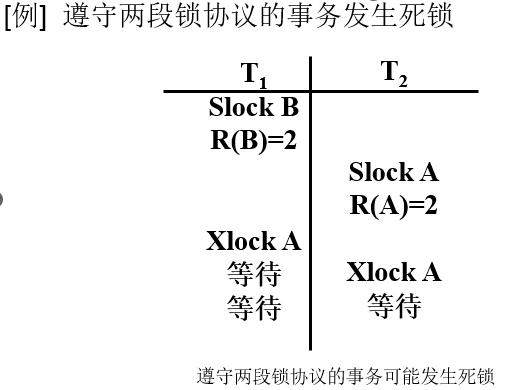
若并发事务都遵守两段锁协议，则对这些事务的任何并发调度策略都是可串行化的

若并发事务的一个调度是可串行化的，不一定所有事务都符合两段锁协议

3.两段锁协议与防止死锁的一次封锁法

一次封锁法要求每个事务必须一次将所有要使用的数据全部加锁，否则就不能继续执行，因此一次封锁法遵守两段锁协议

但是两段锁协议并不要求事务必须一次将所有要使用的数据全部加锁，因此遵守两段锁协议的事务可能发生死锁



(六)封锁的粒度

0.定义

封锁对象的大小称为封锁粒度

封锁的对象：逻辑单元，物理单元

a.在关系数据库中，封锁对象：

逻辑单元: 属性值、属性值集合、元组、关系、索引项、整个索引、整个数据库等

物理单元：页（数据页或索引页）、物理记录等

b.封锁粒度与系统的并发度和并发控制的开销密切相关

封锁的粒度越大，数据库所能够封锁的数据单元就越少，并发度就越小，系统开销也越小；

封锁的粒度越小，并发度较高，但系统开销也就越大

例：若封锁粒度是数据页，事务T1需要修改元组L1，则T1必须对包含L1的整个数据页A加锁。如果T1对A加锁后事务T2要修改A中元组L2，则T2被迫等待，直到T1释放A。

如果封锁粒度是元组，则T1和T2可以同时对L1和L2加锁，不需要互相等待，提高了系统的并行度。

又如，事务T需要读取整个表，若封锁粒度是元组，T必须对表中的每一个元组加锁，开销极大

c.多粒度封锁(Multiple Granularity Locking)

在一个系统中同时支持多种封锁粒度供不同的事务选择

d.选择封锁粒度

同时考虑封锁开销和并发度两个因素，适当选择封锁粒度

d.1需要处理多个关系的大量元组的用户事务：以数据库为封锁单位

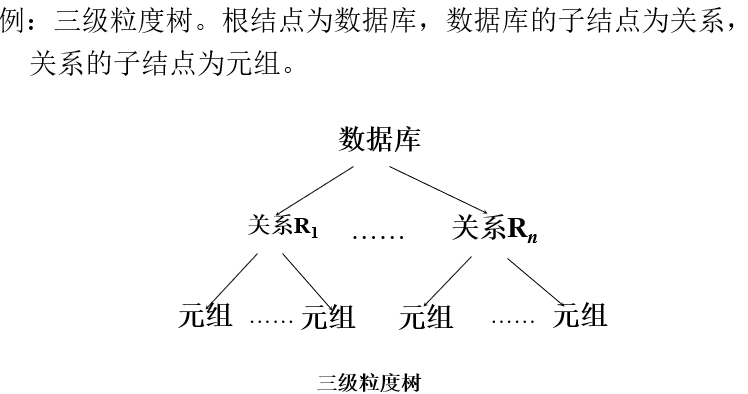
d.2需要处理大量元组的用户事务：以关系为封锁单元

d.3只处理少量元组的用户事务：以元组为封锁单位

1. 多粒度树

a.定义

以树形结构来表示多级封锁粒度。根结点是整个数据库，表示最大的数据粒度。叶结点表示最小的数据粒度



b.多粒度封锁协议

允许多粒度树中的每个结点被独立地加锁，对一个结点加锁意味着这个结点的所有后裔结点也被加以同样类型的锁，在多粒度封锁中一个数据对象可能以两种方式封锁：显式封锁和隐式封锁

c. 显式封锁和隐式封锁

显式封锁: 直接加到数据对象上的封锁

隐式封锁: 该数据对象没有独立加锁，是由于其上级结点加锁而使该数据对象加上了锁

显式封锁和隐式封锁的效果是一样的

c.1系统检查封锁冲突时

要检查显式封锁，还要检查隐式封锁

例如事务T要对关系R1加X锁，系统必须搜索其上级结点数据库、关系R1；还要搜索R1的下级结点，即R1中的每一个元组；如果其中某一个数据对象已经加了不相容锁，则T必须等待

c.2对某个数据对象加锁，系统要检查：

该数据对象有无显式封锁与之冲突

所有上级结点，检查本事务的显式封锁是否与该数据对象上的隐式封锁冲突：(由上级结点已加的封锁造成的）

所有下级结点，看上面的显式封锁是否与本事务的隐式封锁（将加到下级结点的封锁）冲突

2.意向锁

a.引入

引进意向锁（intention lock）目的：提高对某个数据对象加锁时系统的检查效率 如果对一个结点加意向锁，则说明该结点的下层结点正在被加锁

对任一结点加基本锁，必须先对它的上层结点加意向锁

例如，对任一元组加锁时，必须先对它所在的数据库和关系加意向锁

b.意向锁的种类

意向共享锁(Intent Share Lock，简称IS锁)

意向排它锁(Intent Exclusive Lock，简称IX锁)

共享意向排它锁(Share Intent Exclusive Lock，简称SIX锁)

b.1.IS锁

如果对一个数据对象加IS锁，表示它的后裔结点拟（意向）加S锁。

例如：事务T1要对R1中某个元组加S锁，则要首先对关系R1和数据库加IS锁

b.2.IX锁

如果对一个数据对象加IX锁，表示它的后裔结点拟（意向）加X锁。

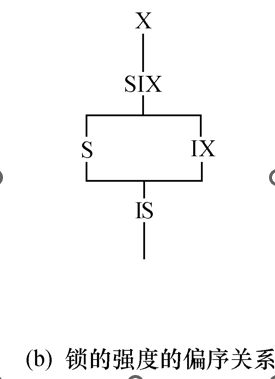
例如：事务T1要对R1中某个元组加X锁，则要首先对关 系R1和数据库加IX锁

b.3.SIX锁

如果对一个数据对象加SIX锁，表示对它加S锁，再加IX锁，即SIX = S + IX。

例：对某个表加SIX锁，则表示该事务要读整个表（所以要对该表加S锁），同时会更新个别元组（所以要对该表加IX锁）。



c.锁的强度

锁的强度是指它对其他锁的排斥程度

一个事务在申请封锁时以强锁代替弱锁是安全的，反之则不然

d.具有意向锁的多粒度封锁方法

申请封锁时应该按自上而下的次序进行

释放封锁时则应该按自下而上的次序进行

例如：事务T1要对关系R1加S锁

要首先对数据库加IS锁，检查数据库和R1是否已加了不相容的锁(X或IX)，不再需要搜索和检查R1中的元组是否加了不相容的锁(X锁)

e.具有意向锁的多粒度封锁方法

提高了系统的并发度，减少了加锁和解锁的开销，在实际的数据库管理系统产品中得到广泛应用