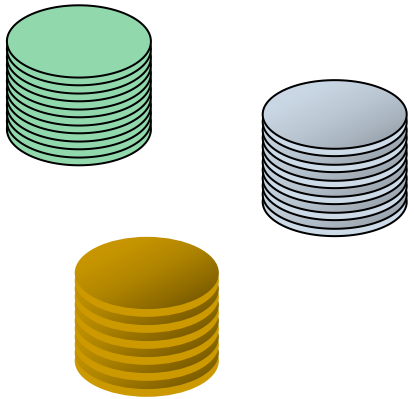
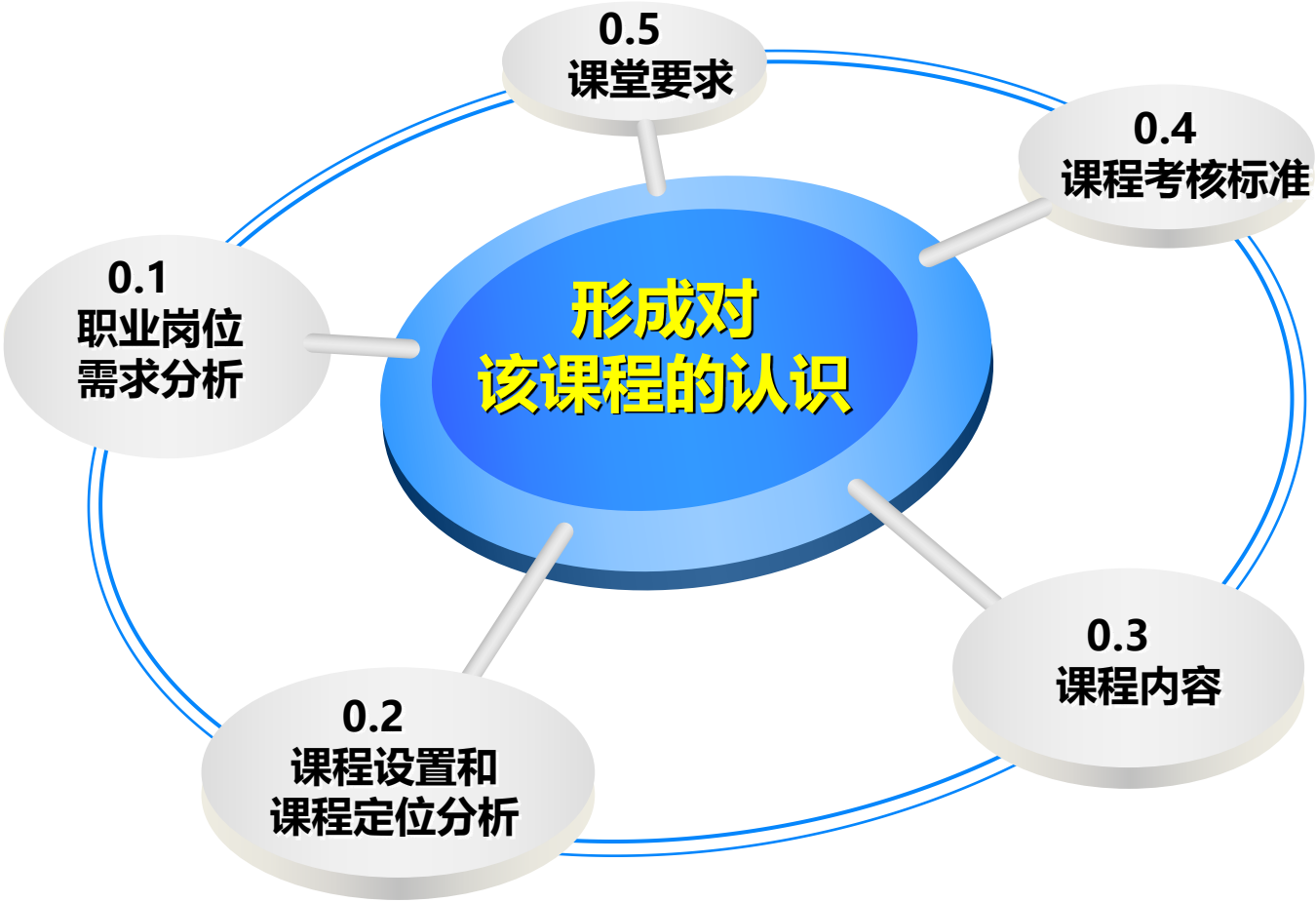


数据库系统

北京工业大学耿丹学院
计算机科学与技术专业

本讲概述


0.5
课堂要求
0.4
课程考核标准
0.1
职业岗位
需求分析
形成对
该课程的认识
0.3
课程内容
0.2
课程设置和
课程定位分析

0.1 职业岗位需求分析

通过对以下网站进行访问：

①智联招聘网（**http://www.zhaopin.com**）；

②前程无忧网（**http://www.51job.com**）；

③中华英才网（ **https://www.chinahr.com** ）；

对招聘信息中涉及本专业的相关岗位如：数据库工程师、数据库管理员和数据库系统开发、软件行业的软件开发工程师等进行分析，把与本课程相关的信息进行汇总，比如涉及到数据库的基本知识、操作技能和基本素质的要求等等进行汇总。

0.1 职业岗位需求分析

数据库工程师

岗位要求：

1. 计算机及其相关专业本科及以上学历；
2. 掌握数据库技术的基本概念、原理、方法和技术；
3. 能够使用SQL语言实现数据库操作；
4. 具备数据库系统安装、配置及数据库管理与维护的基本技能；
5. 了解数据库应用系统的生命周期及其设计、开发过程；
6. 熟悉常用的数据库管理和开发工具，具备用指定的工具管理和开发简单数据库应用系统的能力；
7. 了解数据库技术的最新发展；
8. 具有严谨认真的工作态度，良好的沟通能力、团队合作能力

0.1 职业岗位需求分析

数据库管理员(DBA)

岗位要求：

1. 熟悉SQL数据库技术，熟练掌握T-SQL语言，能开发编写数据库管理、sql脚本；
2. 精通数据库管理维护，熟练编写复杂存储过程，函数，触发器，并能进行性能优化；
3. 能设计大型数据库结构，撰写规范的技术文档；
4. 熟悉数据库的安装部署、管理、备份、恢复与故障处理；有较强的SQL编程经验，有数据库调优方面的经验；
5. 熟悉SQL Server、MySQL、Oracle、Sybase、DB2等主流数据库；
6. 思维清晰敏捷,逻辑分析能力强,良好的口头和书面表达能力,善于与人沟通
7. 有责任意识，工作认真细致，服从工作安排，学习新技术能力强，良好的技术文档编写及表达能力及沟通协调能力。
8. 善于学习，具有独立解决问题能力,认真负责，责任心强，能主动承担工作
9. 良好的沟通能力、团队合作，具有很好的编写文档能力

0.1 职业岗位需求分析

数据库开发工程师

岗位要求：

1. 在云计算/数据库/分布式系统/存储相关领域有3年以上开发经验，具有丰富的服务器后端架构设计和研发经验；

2. 对数据库系统的构架和开发具有较为全面和深入的理解；

4. 熟悉数据库日常管理维护；精通T-SQL、存储过程、SQL优化等；

5. 熟悉各类数据库如MySQL、MongoDB、PostgreSQL、Redis其中之一，有MySQL源码开发经验者优先；

6. 对数据库业界最新动态时刻关注，探索RDBMS/NoSQL/HTAP/NewSQL等技术实现和应用场景；

7. 具有团队协作精神，较强的逻辑分析及沟通表达能力，善于学习，勇于探索新领域

0.1 职业岗位需求分析

软件开发工程师

岗位要求：

1．熟悉C/C++、VC++、C#、Java、.NET,Python,PHP等编程语言，及相关网络协议；

2．精通数据库管理、面对对象的分析和设计方法、关系型数据库结构设计与编程、设计模式；

3．熟悉Sql Server、Oracle、MySql、Access两项以上，并有存储过程、触发器等开发经验；

4．熟悉常见的建模工具比较（PowerDesigner、Rose、Visio、BpWin、ErWin一项以上；

5．计算机专业基础知识扎实（专业知识为：软件开发流程、数据库原理、面向对象原理、计算机原理等）。

6．参与需求分析，业务功能的分解、设计，有一定的项目规划和管理能力；

7.能承受工作压力，工作认真、踏实，责任心强。

8．具有团队合作精神，善于与他人沟通与合作。

0.1 职业岗位需求分析

软件行业从业人员能力要求（1）

- 软件开发工具

① 熟悉或精通C#、Java、Python、PHP、VC等开发工具的一种或几种。

② 熟悉ASP.NET、ASP、JSP等网络编程技术的一种或几种。

③ 熟悉Windows平台下的程序开发，了解Linux等开发平台。

④ 熟练使用ADO.NET实现数据库访问的操作。

⑤ 熟悉JavaScript、VUE、React等前端开发工具的一种或几种。

0.1 职业岗位需求分析

软件行业从业人员能力要求（2）

- 数据库设计、管理

① 熟悉或精通Access、Microsoft SQL Server、Oracle、DB2、MySQL等主流数据库管理系统的一种或几种。

② 了解Sqlite、PostGRE、BerkleyDB等嵌入式数据库管理系统。

③ 了解数据库理论及开发技术，了解数据库建模，熟悉常用数据库建模工具。

④ 精通T-SQL 或PL/SQL、存储过程和触发器、SQL优化及数据库管理，能够快速解决数据库的故障。

⑤ 熟悉SQL的设计和开发(包括表设计和优化，复杂查询语句的调试和优化)。

⑥ 熟悉数据库后台管理和SQL编程。

软件行业从业人员能力要求（3）

- 基本素质和工作态度

① 积极的工作态度和较强的责任心，良好的沟通和学习能力。

② 具有主观能动性、团队合作精神和强烈的事业心。

③ 较强的敬业精神，创新精神，开拓意识及自我规范能力。

④ 强烈的客户服务意识、较强的理解能力，能够在压力下独立完成工作。

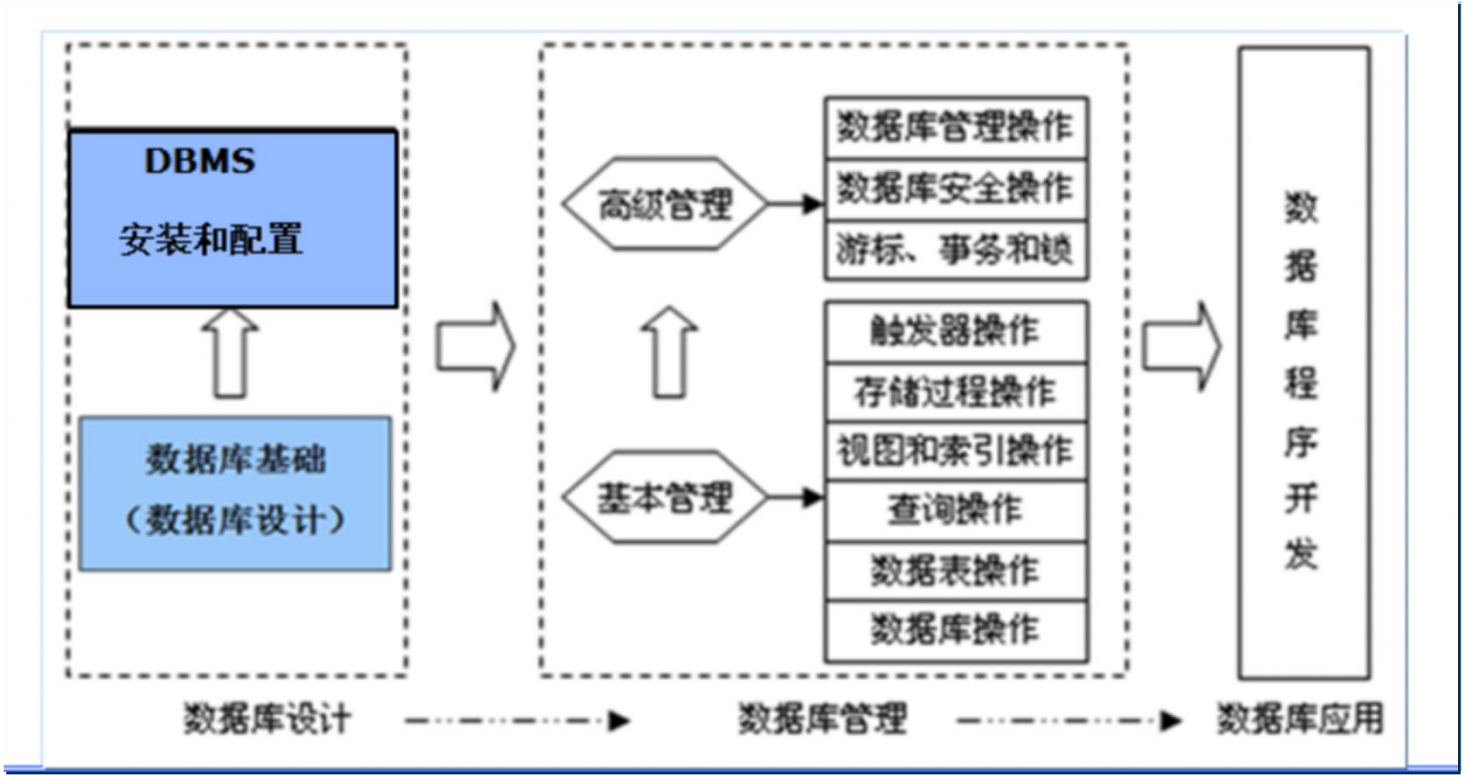
0.2 课程设置和课程定位分析

	大一		大二		大三		大四	
	第一学期	第二学期	第三学期	第四学期	第五学期	第六学期	第七学期	第八学期
专业必修课	专业认知与职业生涯	高等数学II	大学物理	概率论与数理统计	操作系统	计算机组成原理与体系结构	毕业实习（第7或第8学期）	毕业设计（论文）
	高等数学I	面向对象程序设计	数据库系统	计算机网络	Web应用开发技术	编译原理		
	线性代数	离散数学	数据结构与算法	前端综合技术	电子技术基础	软件工程		
	高级语言程序设计			Python编程与实践	专业英语与信息检索	人工智能		
	网页设计基础			认知实习	计算机网络安全	大数据开发技术		

0.2 课程设置和课程定位分析

教学目的	(1)理解数据库的基本概念和数据库设计原理 (2)熟练掌握SQL的使用 (3)熟练掌握管理数据库、表和视图的操作 (4)熟练掌握管理数据完整性和索引的操作 (5)熟练掌握SQL语言的常用语法与应用 (6)熟练掌握管理存储过程、存储函数与触发器的操作 (7)掌握对用户、角色和管理 (8)熟练掌握数据库备份与恢复的操作
教学重点	(1)管理数据库、表和视图 (2)管理数据完整性和索引 (3)SQL语言的常用语法与应用 (4)管理存储过程、存储函数与触发器 (5)数据库备份与恢复

0.3 课程内容


DBMS
安装和配置
数据库基础
（数据库设计）
高级管理
基本管理
数据库管理操作
数据库安全操作
游标、事务和锁
触发器操作
存储过程操作
视图和索引操作
查询操作
数据表操作
数据库操作
数据库程序开发
数据库设计
数据库管理
数据库应用

0.3 课程内容


建模工具：
Power design
Workbench
第一阶段
掌握数据库设计的基本原理及基本步骤
SQL 语言
第二阶段
掌握数据库基本操作
第三阶段
掌握DBA的基本技能
数据库日常维护；
存储过程的编写等。
第四阶段
掌握数据库应用系统的开发

0.4 课程考核说明

成绩分配比例说明

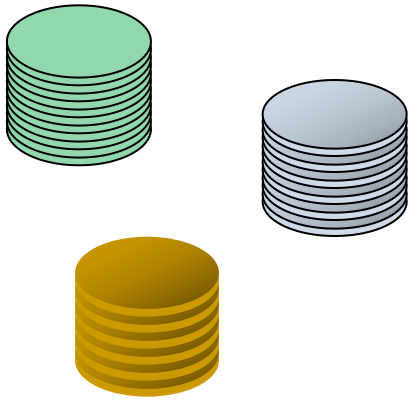
- 平时成绩（60%）
  - 平时考勤（10%）
  - 课堂表现（20%）
  - 实验作业（30%）
  - 机试作业（40%）
- 期末成绩（40%）
  - 机试（50%）
  - 综合大作业（50%）

0.5 课堂要求

课堂要求

- 固定座位
- 带书，本，笔，有电脑的可自带电脑
- 不允许迟到早退
- 不允许在课堂上接听电话、玩游戏，随意走动
- 不允许将食物带入教室

若出现上述情况之一，扣平时成绩



数据库系统

第一章 绪论

北京工业大学耿丹学院
计算机科学与技术专业

本讲概述


1.3
小结
掌握数据库
基本概念及
数据库系统模型
1.1
数据库系统概述
数据
数据库
数据管理系统
数据库系统
1.2
数据库系统结构
三级模式
二级映像

思考：

我们每天都在自觉或不自觉地与数据库打交道；

学校教务系统、移动电话记录、商品采购等等

数据库中存储了哪些内容？？

从数据库中能检索到哪些信息？？

1.1 数据库系统概述

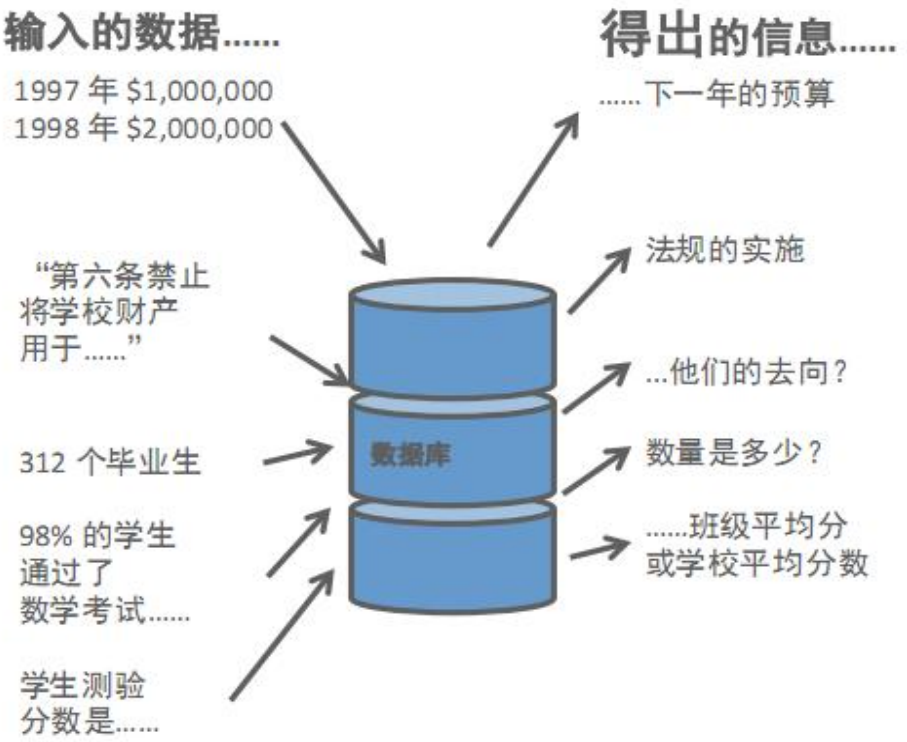
1、数据和信息

- 数据是原始资料或未处理的资料
- 信息是知识、情报、有特定意义或作用的一项特定数据。
- 信息通常是对数据进行组合、比较分析或计算的结果。

1.1 数据库系统概述

1、数据和信息

- 以测验成绩为例。
- 如果一个班级中的每名学生都得到一个分数，则通过这些分数可以计算出班级平均分数。
- 然后通过班级平均分数又可以计算出学校的平均分数。



1.1 数据库系统概述

1.1.1 基本概念

1、数据(Data)

- 数据是数据库中存储的基本对象
- 数据的定义——描述事物的符号记录。
- 数据的种类——数字、文字、图形、图象、声音等。

问题：有了数据的表现形式是不是就可以完全表达其内容了呢？

例如：数据：**93**

表达的内容可以是：学生某门课程的成绩

专业总人数

某个人的体重等等

- 数据的解释——对数据含义的说明，数据的含义称为数据的语义。
- 数据的特点——数据与其语义是不可分的。

1.1 数据库系统概述

1.1.1 基本概念

1、数据(Data)

数据举例

- 学生档案中的记录：（李明，男，2004年7月 ，江苏，信息工程学院，2022）
- 数据的形式不能完全表达其内容
- 数据的解释

  语义：学生姓名、性别、出生年月、籍贯、所在学院、入学时间

  解释：李明，男，2004年7月出生，江苏人，2022年考入信息工程学院

2、数据库(Database,简称DB)

- 定义
  - 是长期存储在计算机上，有组织、可共享的大量数据的集合。
- 特点
  - 数据按一定的数据模型组织、描述和储存；
  - 可为各种用户共享；
  - 冗余度较小；
  - 独立性较高；
  - 易扩展。
  - …

1.1 数据库系统概述

❖ 数据库举例

表 1.1 图书表

ISBN 号	书名	作者	出版社	出版日期
9787302189268	数据库原理与应用（Access）	张巍，曹起武	清华大学出版社	2009 年 1 月
9787302181996	数据库技术应用基础	史九林，窦显玉	清华大学出版社	2009 年 1 月
9787302182580	网络数据库原理与应用	刘翔	清华大学出版社	2008 年 10 月
9787121075681	数据库实用教程	郑阿奇	电子工业出版社	2009 年 1 月
9787111203117	数据库原理及应用	李辉	机械工业出版社	2007 年 1 月
9787040195835	数据库系统概论（第 4 版)	王珊，萨师煊	高等教育出版社	2006 年 5 月

有名读者需要查询清华大学出版社在2009年出版的、书名包含“数据库”的图书，查询结果如下表所示。

ISBN 号	书名	作者	出版社	出版日期
9787302189268	数据库原理与应用（Access）	张巍，曹起武	清华大学出版社	2009 年 1 月
9787302181996	数据库技术应用基础	史九林，窦显玉	清华大学出版社	2009 年 1 月

1.1 数据库系统概述

表1.2 读者表

借阅证号	姓名	单位	联系电话
201804070501	张勇	北京工业大学耿丹学院工学院	010-60411112
201804070502	李明	北京工业大学耿丹学院工学院	010-60411113
201804070503	王强	北京工业大学耿丹学院工学院	010-60411114
201804070504	刘明	北京工业大学耿丹学院工学院	010-60411115

表1.3 借阅表

ISBN号	借阅证号	借阅日期	应还日期
9787111203117	201804070501	2018/9/10	2018/12/10
9787121075681	201804070502	2018/9/11	2018/12/11
9787302189268	201804070501	2018/9/10	2018/12/10
9787302189268	201804070502	2018/9/13	2018/12/13

张勇想要查询一下自己借的书归还日期

查询结果

姓名	借阅证号	ISBN号	借阅日期	应还日期
张勇	201804070501	9787111203117	2018/9/10	2018/12/10
张勇	201804070501	9787302189268	2018/9/10	2018/12/10

如何科学地组织和存储数据？

如何高效地获取和维护数据？

3、数据库管理系统（Database Management System，简称DBMS）

- **DBMS**
  - 数据库管理系统是位于用户与操作系统之间的一层数据管理软件。

1.1 数据库系统概述

❖ DBMS的主要功能

- 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
- 数据操纵功能： 提供数据操纵语言(DML）——实现对数据库的基本操作(查询、插入、删除和修改)
- 数据库事务管理和运行功能： 并发控制，恢复机制
- 数据库的建立和维护功能： 数据的输入、转换功能、数据库转储、介质故障恢复、数据库的重组织、性能监视等

问题：用户是用DML还是DDL完成下列任务？

a.改变客户地址 DML

b.定义目录表 DDL

c.输入一个新员工的信息 DML

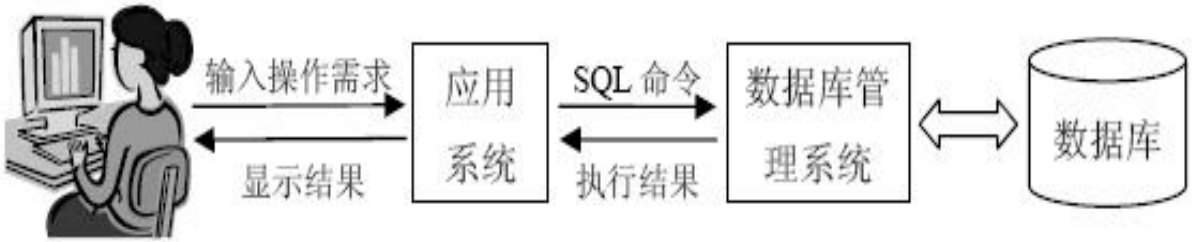
4、数据库系统

❖ 什么是数据库系统

- 数据库系统（Database System，简称DBS）是指在计算机系统中引入数据库后的系统。

❖ 数据库系统的构成

由数据库、DBMS、应用程序、用户构成。



- **描述DBS，DBMS，DB，Data之间的关系**

1.1 数据库系统概述

1.1.2 数据管理技术的产生和发展

- 什么是数据管理

  对数据进行分类、组织、编码、存储、检索和维护，是数据处理的中心问题。

  数据处理是指对各种数据进行收集、存储、加工和传播的一系列活动的总和。

- 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
- 数据管理技术的发展过程
  - 人工管理阶段(50年代中期以前)
  - 文件系统阶段(50年代末--60年代中)
  - 数据库系统阶段(60年代末--现在)
- 未来发展趋势
  - 面向云计算的云数据管理技术（如：Google的GFS，BigTable，Amazon的Dynamo）

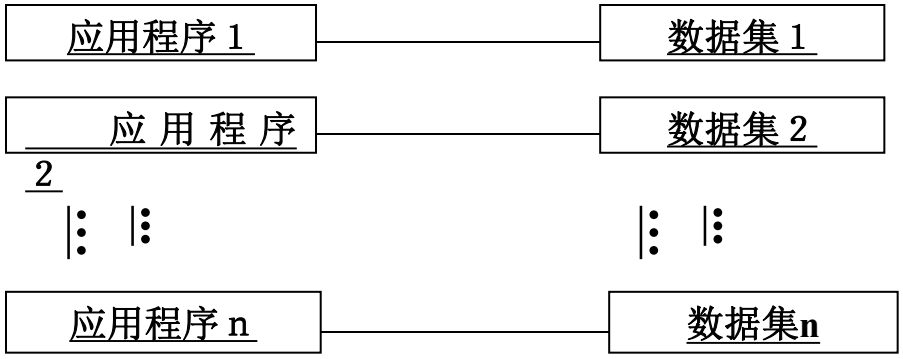
1.1 数据库系统概述

一、人工管理阶段

- 时期：50年代中期以前
- 产生的背景
  - 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：没有操作系统和管理软件

1.1 数据库系统概述

一、人工管理阶段



- 特点
  - 数据不保存
  - 数据的管理由应用程序完成：

    没有相应的软件系统专门负责数据的管理工作。
  - 数据冗余度大且不共享：当多个应用程序涉及某些相同的数据时，必须由各自的应用程序分别定义和管理这些数据，无法共享利用，因此存在大量冗余数据。
  - 数据不具有独立性：程序依赖于数据，如果数据的类型、格式、或输入输出方式等逻辑结构或物理结构发生变化，必须对应用程序做出相应的修改。

二、文件系统阶段

- 时期
  - 50年代末--60年代中
- 产生的背景
  - 应用需求　科学计算、管理
  - 硬件水平　磁盘、磁鼓
  - 软件水平　有了操作系统和专门的数据管理软件

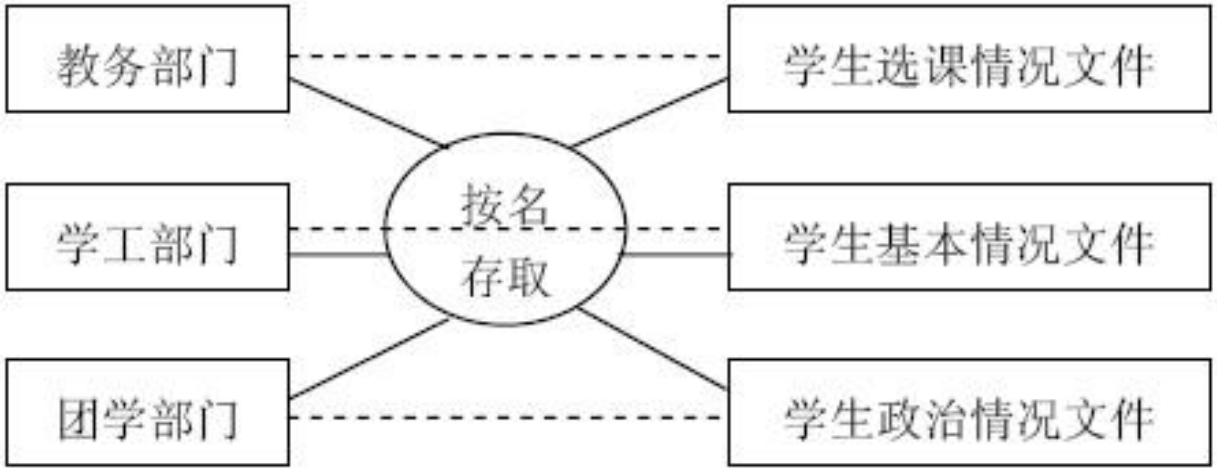
二、文件系统阶段(续)

❖特点

- 数据可长期保存：数据以文件形式保存，用户可随时对文件进行查询、修改和增删等处理。
- 数据由文件系统进行管理：程序员只与文件名打交道，不必明确数据的物理存储，大大减轻了程序员的负担。
- 程序与数据间有一定独立性：由专门的软件即文件系统进行数据管理，程序和数据间由软件提供的存取方法进行转换，数据存储发生变化不一定影响程序的运行。

二、文件系统阶段(续)

- 例如：用文件系统管理学生数据



文件系统阶段

二、文件系统阶段(续)

与人工管理阶段相比，文件系统阶段对数据的管理有了很大的进步，但一些根本性问题仍没有彻底解决，主要表现在以下三方面：

- 数据冗余度大：各数据文件之间没有有机的联系，一个文件基本上对应于一个应用程序，数据不能共享。
- 数据独立性低：数据和程序相互依赖，一旦改变数据的逻辑结构，必须修改相应的应用程序。而应用程序发生变化，如改用另一种程序设计语言来编写程序，也需修改数据结构。
- 数据一致性差：由于相同数据的重复存储、各自管理，在进行更新操作时，容易造成数据的不一致性。

三、数据库系统阶段

- 时期
  - 60年代末以来
- 产生的背景
  - 应用背景　　大规模管理
  - 硬件背景　　大容量磁盘
  - 软件背景　　有数据库管理系统

三、数据库系统阶段

- 主要特点：
- 数据结构化
  - 数据库管理系统实现数据的**整体结构化**，这是数据库的主要特征之一，也是数据库管理系统与文件系统的本质区别
    - 数据不仅仅是内部结构化，而是将数据以及数据之间的联系统一管理起来，使之结构化。

学生文件Student的记录结构

学号	姓名	性别	出生日期	所学专业	家庭住址	联系电话

课程文件Course的记录结构

课程号	课程名称	学时	学分	教材名称

学生成绩文件Score的记录结构

学号	课程号	学期	成绩

学生、课程、学生成绩文件结构

1.1 数据库系统概述

- 在数据库中的数据不是仅仅针对某一个应用，而是面向全组织的所有应用。
- 例如，一个学校的信息系统中不仅要考虑教务处的学生成绩管理，还要考虑学工处的学籍注册管理、学生奖惩管理、学生家庭成员管理，以及财务处的学生缴费管理;同时还要考虑科研处的科研管理、人事处的教职工人事管理和工资管理等。
- 因此，学校信息系统中的学生数据要面向全校各个职能管理部门和院系的应用，而不仅仅是教务处的一个学生成绩管理应用。

1.1 数据库系统概述

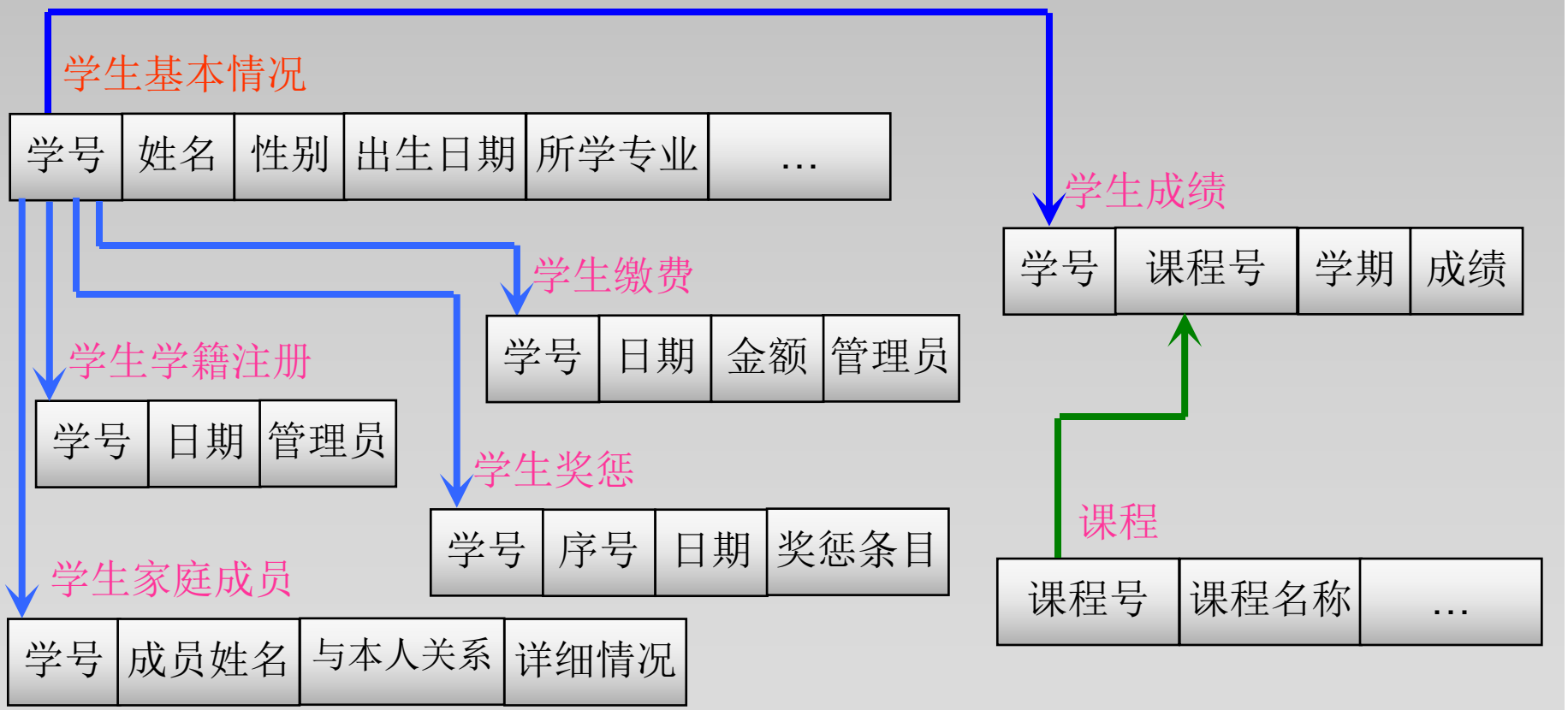


图1-4 某校信息管理系统中的学生数据

1.1 数据库系统概述

❖ 数据库管理系统的主要特点

- **数据的共享度高，冗余度底，易扩充**
  - 数据库管理系统从整体角度描述和组织数据，数据不再是面向某个应用，而是面向整个系统
  - 数据可以被多个用户、多个应用共享使用
  - 数据共享可以大大减少数据的冗余，避免数据之间的不一致性
- **数据独立性高**
  - 数据独立是指数据的使用(即应用程序)与数据的说明(即数据的组织结构与存储方式)分离
    - 这样，应用程序只需要考虑如何使用数据，而无须关心数据库中的数据是如何构造和存储的
    - 因而，各方(在一定范围内)的变更互不影响

1.1 数据库系统概述

- 数据库管理系统的主要特点
  - 数据独立性高
    - 数据独立性用来描述应用程序与数据结构之间的依赖程度，包括数据的物理独立性和数据的逻辑独立性，依赖程度越低则独立性越高
    - 物理独立性是指用户的应用程序与数据库中数据的物理结构是相互独立的。数据在磁盘上如何组织和存储由DBMS负责，应用程序只关心数据的逻辑结构；当数据的物理存储结构改变时，应用程序不用修改
    - 逻辑独立性是指用户的应用程序与数据库中数据的逻辑结构是相互独立的。数据的(全局)逻辑结构由DBMS负责，应用程序只关心数据的局部逻辑结构(即应用视图)，数据的(全局)逻辑结构改变了，应用程序也可以不用修改 。

三、数据库系统阶段

- 特点：

- **数据由DBMS统一的管理和控制**

数据的安全性（Security）保护

- 使每个用户只能按指定方式使用和处理指定数据，保护数据以防止不合法的使用造成的数据的泄密和破坏。

数据的完整性（Integrity）检查

- 将数据控制在有效的范围内，或保证数据之间满足一定的关系。

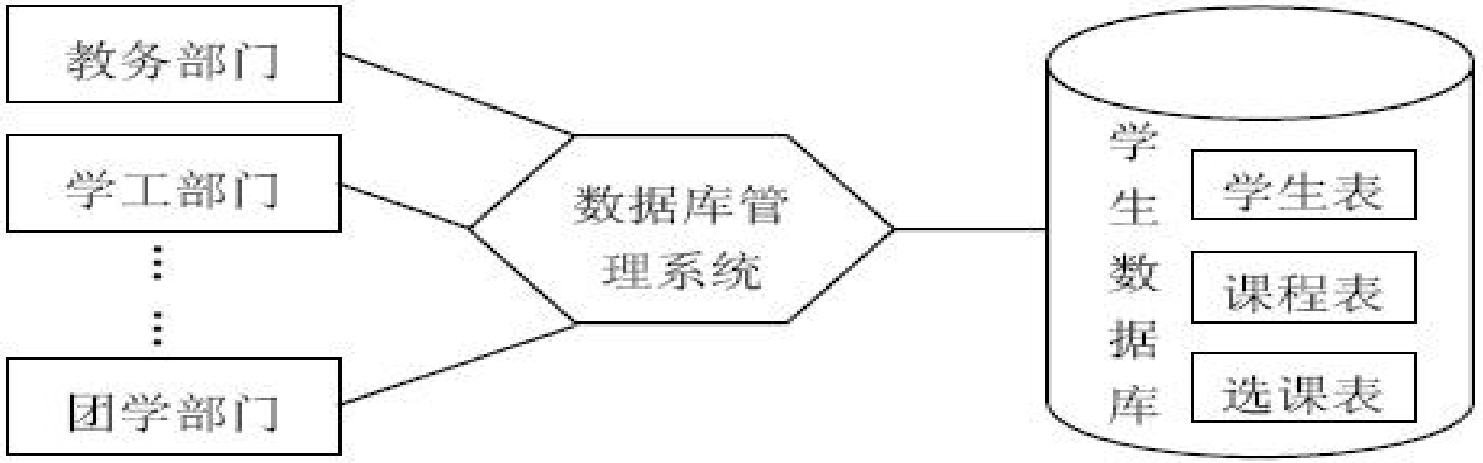
并发（Concurrency）控制

- 对多用户的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（Recovery）

- 将数据库从错误状态恢复到某一已知的正确状态。

1.1 数据库系统概述



数据库系统阶段

应用程序与数据之间的关系

1.2 数据库系统结构

- DBMS：隐藏关于数据存储和维护的某些细节，为用户提供数据在不同层次上的视图，即**数据抽象**，方便不同的使用者可以从不同的角度去观察和利用数据库中的数据。
- 物理层抽象
  - 最低层次的抽象，描述数据实际上是怎样存储的
- 逻辑层抽象
  - 描述数据库中存储什么数据以及这些数据之间存在什么关联
  - 提供给数据库管理员和数据库应用开发人员使用的，必须明确知道数据库中应该保存哪些信息
- 视图层抽象
  - 最高层次的抽象，只描述整个数据库的某个部分，即局部逻辑结构
  - 系统可以为同一数据库提供多个视图，每一个视图对应一个具体的应用，亦称为应用视图

1.2 数据库系统结构

- 根据数据抽象的3个不同级别，DBMS也提供观察数据库的3个不同角度，以方便不同的用户使用数据库的需要。这就是数据库的三级模式结构

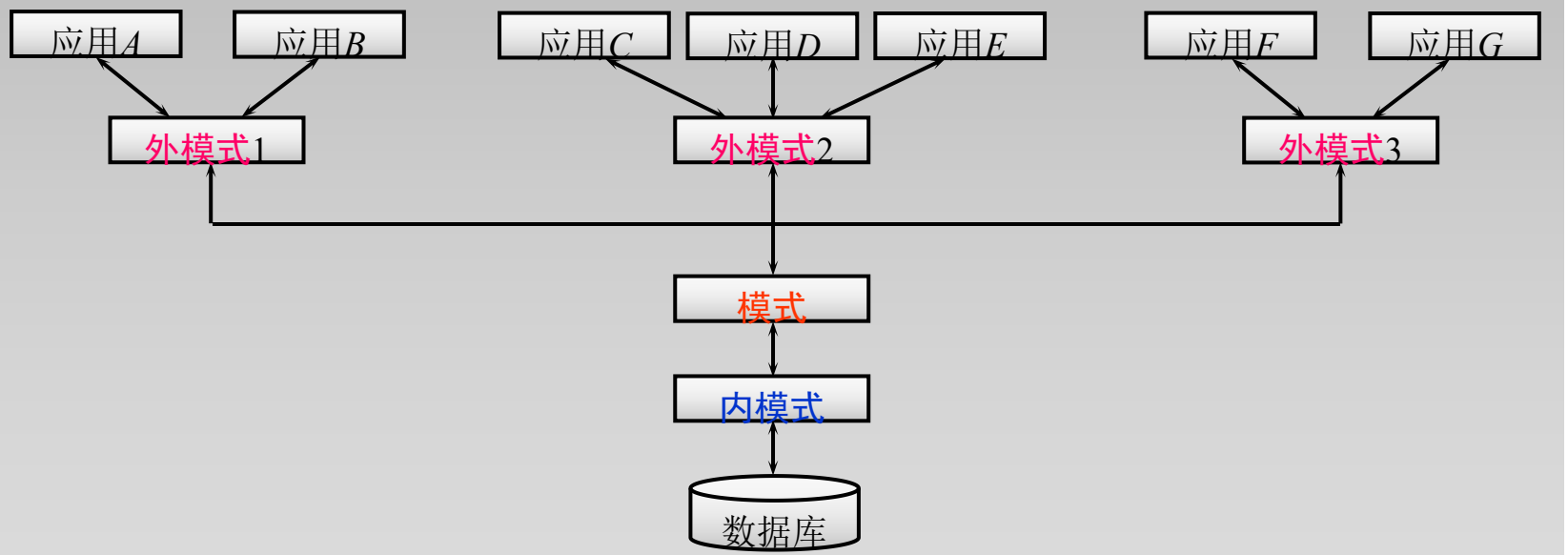


图1-10　数据库的三级模式结构

1.2 数据库系统结构

1. 模式（Schema）

- 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、开发工具及高级程序设计语言无关
- 模式的定义
  - 定义数据的逻辑结构（数据项，数据项的名字，类型，取值范围）
  - 定义数据之间的联系
  - 定义与数据有关的安全性、完整性要求

1.2 数据库系统结构

2. 外模式（External Schema）

- 外模式（也称子模式或用户模式）
  - 数据库用户（应用程序员和最终用户）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
    - 外模式通常是模式的子集；
    - 一个数据库可以有多个外模式。反映了不同的用户的应用需求、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。

3. 内模式（Internal Schema）

- 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式
    - 记录的存储方式（顺序存储，按照B树结构存储，按hash方法存储）
    - 索引的组织方式
    - 数据是否压缩存储
    - 数据是否加密
    - 数据存储记录结构的规定
- 一个数据库只有一个内模式

▪实例

用户访问数据库的过程实例

为简单起见，假设数据库的模式中存在学生表：

stu_info (stu_id, name, birthday, sex, sdept)。

有两个用户共享该学生表：

- 用户/应用1：
  处理的是学生的学号（stu_id）、姓名（name）和性别（sex）数据；
- 用户/应用2：
  处理的是学生的学号（stu_id）、姓名（name）和所在系（sdept）数据。

由于这两个用户习惯处理中文列名，因此分别为其定义外模式：

- 花名册1（学号，姓名，性别）
- 花名册2（学号，姓名，所在系）

该学生表以链表的结构进行存储。

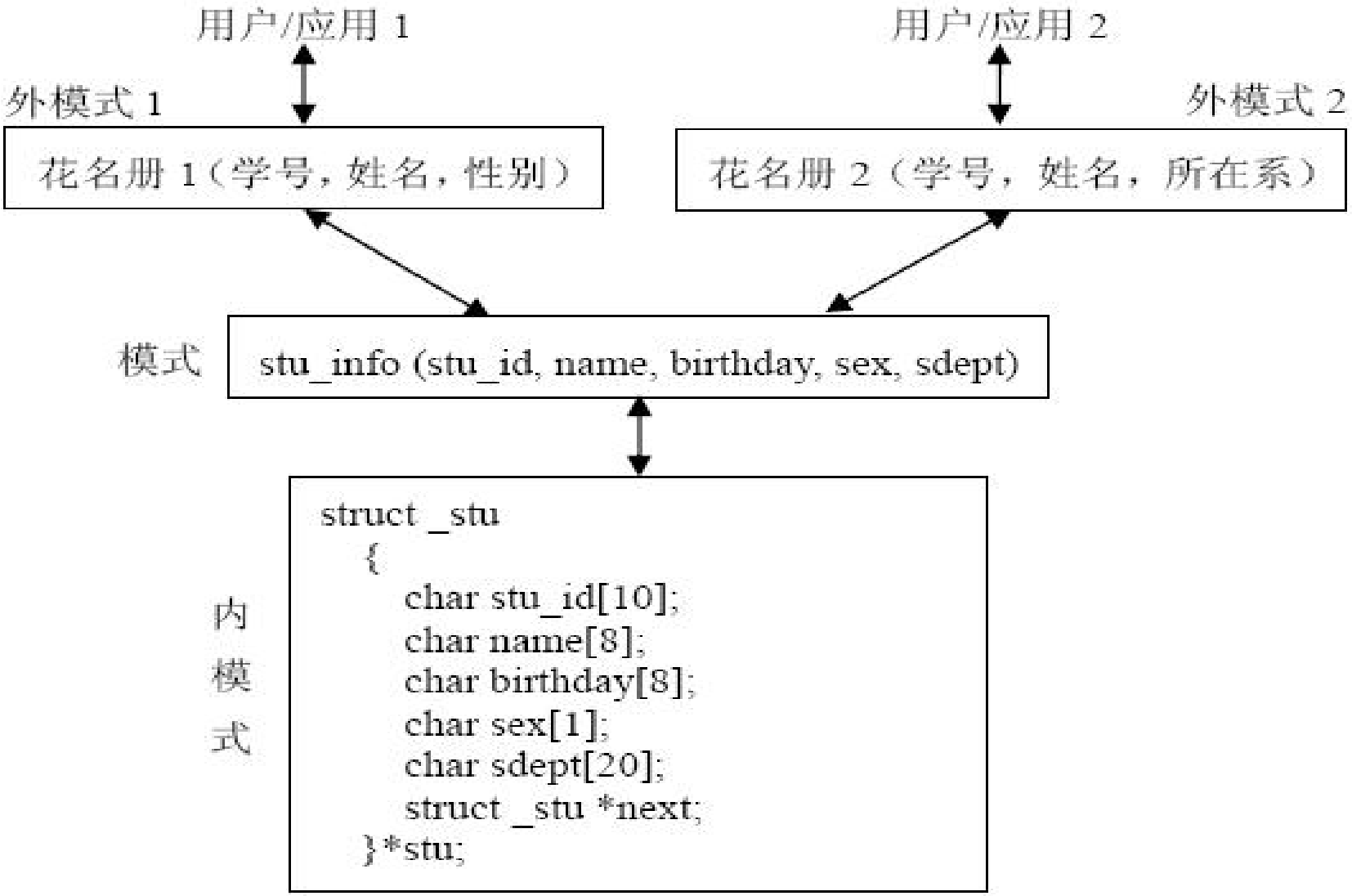


图 1.23 三级模式结构的一个实例

1.2 数据库系统结构

1.2.3 数据库的二级映象功能与数据独立性

1. 外模式 / 模式映象

- 定义外模式与模式之间的对应关系；
- 每一个外模式都对应一个外模式 / 模式映象；
- 映象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 / 模式映象，使外模式保持不变；
- 当数据的总体逻辑结构（模式）发生变化时，由于应用程序是依据数据的外模式编写的，从而应用程序不必修改，保证了数据与程序的逻辑独立性，简称数据的逻辑独立性。

1.2 数据库系统结构

2．模式／内模式映象

- 模式／内模式映象定义了数据全局逻辑结构与存储结构之间的对应关系。例如，说明逻辑记录和字段在内部是如何表示的；
- 数据库中模式／内模式映象是唯一的；
- 该映象定义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／内模式映象，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。

1.2 数据库系统结构

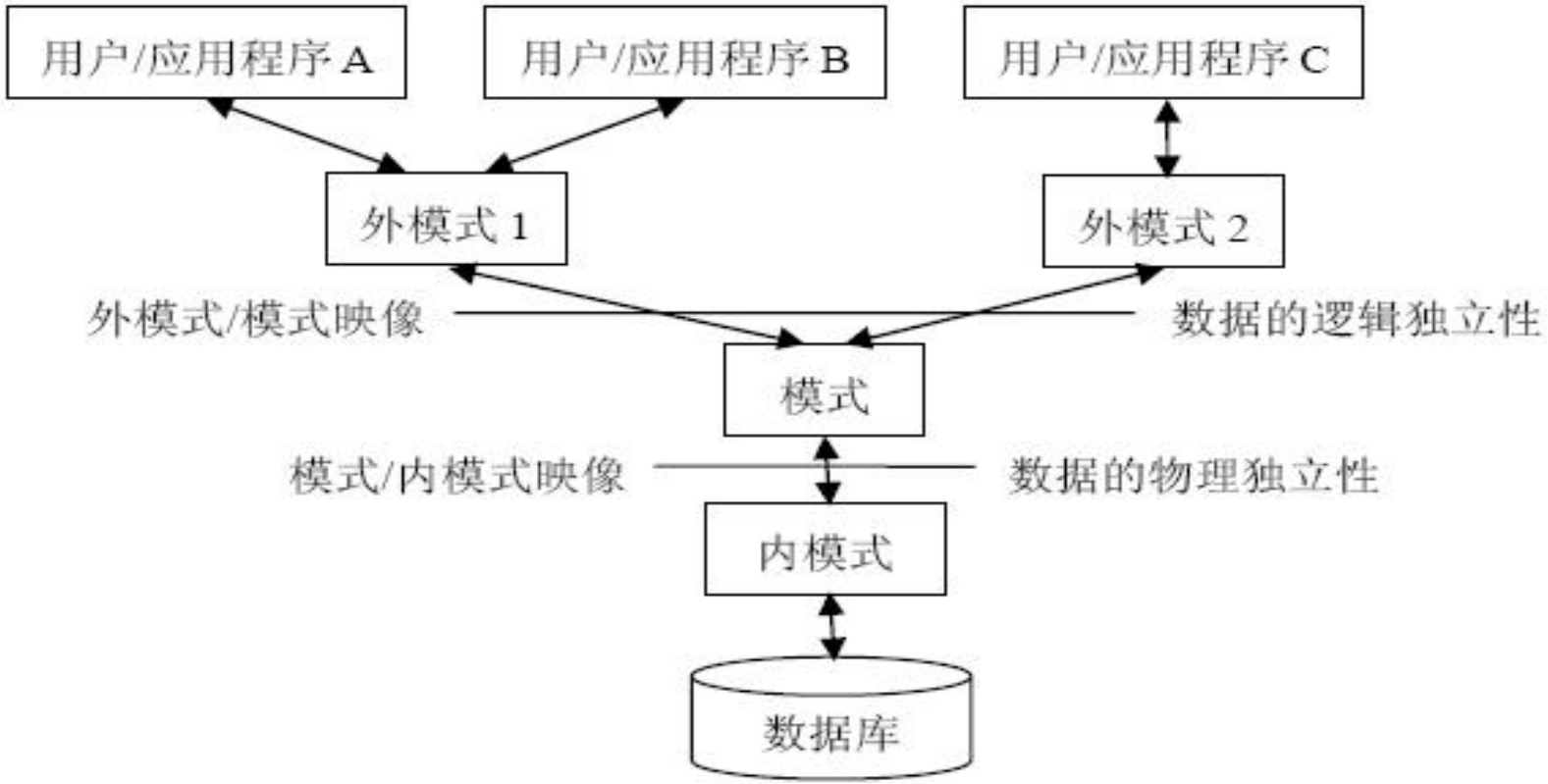


图 1.22 数据库系统的三级模式结构及其两级映像功能

▪实例

用户访问数据库的过程实例

为简单起见，假设数据库的模式中存在学生表：

stu_info (stu_id, name, birthday, sex, sdept)。

有两个用户共享该学生表：

- 用户/应用1：
  处理的是学生的学号（stu_id）、姓名（name）和性别（sex）数据；
- 用户/应用2：
  处理的是学生的学号（stu_id）、姓名（name）和所在系（sdept）数据。

由于这两个用户习惯处理中文列名，因此分别为其定义外模式：

- 花名册1（学号，姓名，性别）
- 花名册2（学号，姓名，所在系）

该学生表以链表的结构进行存储。

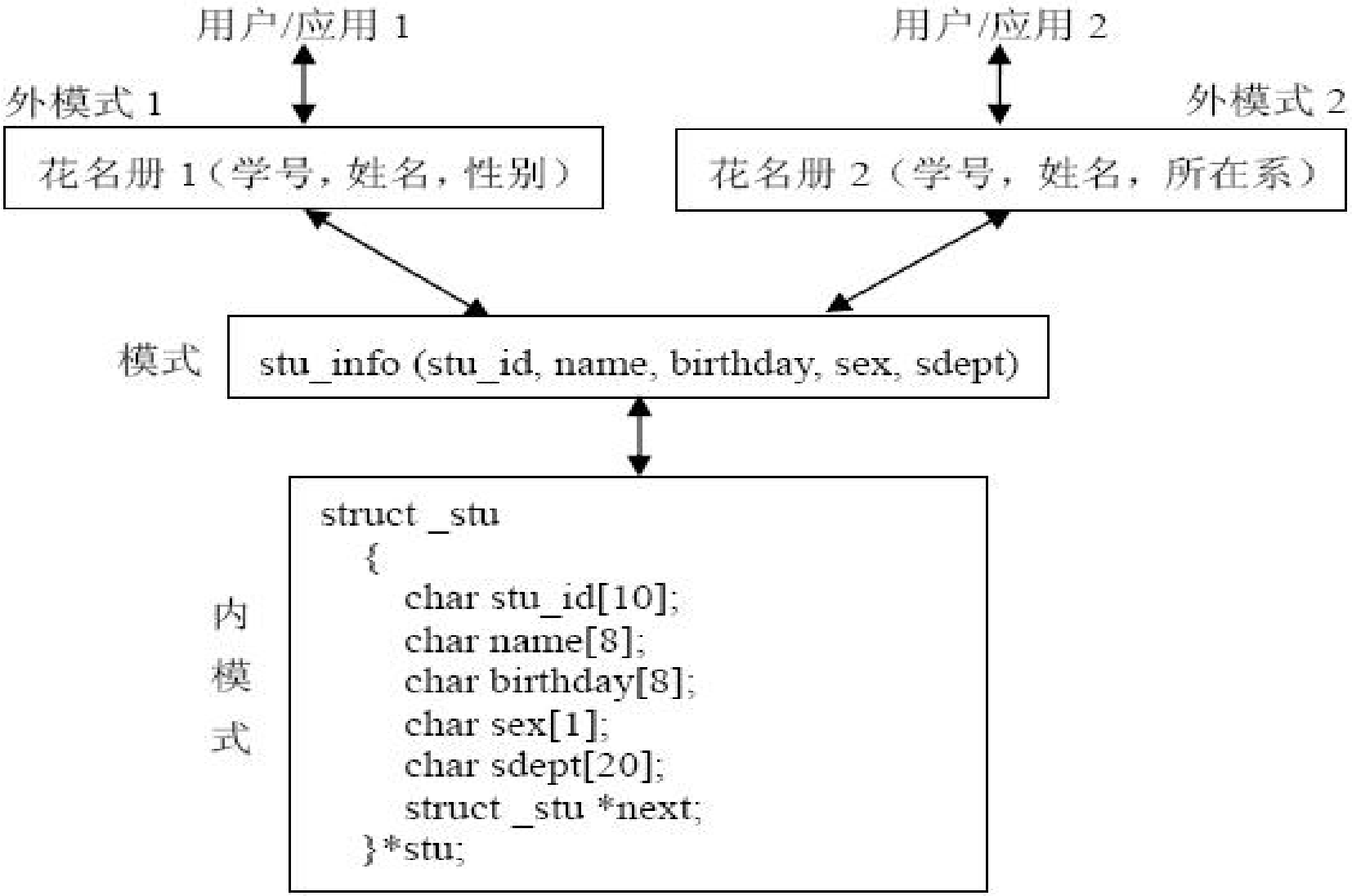


图 1.23 三级模式结构的一个实例

- 外模式/模式映像

花名册1.学号←→stu_info. stu_id

花名册1.姓名←→stu_info. name

花名册1.性别←→stu_info. sex

花名册2.学号←→stu_info. stu_id

花名册2.姓名←→stu_info. name

花名册2.所在系←→stu_info. sdept

- 模式/内模式映像

stu_info. stu_id←→stu->stu_id

stu_info. name←→stu->name

stu_info. birthday←→stu->birthday

stu_info. sex←→stu->sex

stu_info. sdept←→stu->sdept

- 假设数据的逻辑结构发生了变化，例如，将stu_info表一分为二：
  - stu_info1(stu_id, name, birthday, sex)
  - stu_info2(stu_id, name, birthday, sdept)
- 为使外模式1和外模式2不变，进而使相应的应用程序不变，只需将相应的外模式/模式映像分别修改为：

花名册1.学号←→stu_info1. stu_id

花名册1.姓名←→stu_info1. name

花名册1.性别←→stu_info1. sex

花名册2.学号←→stu_info2. stu_id

花名册2.姓名←→stu_info2. name

花名册2.所在系←→stu_info2. sdept

1.3 小结

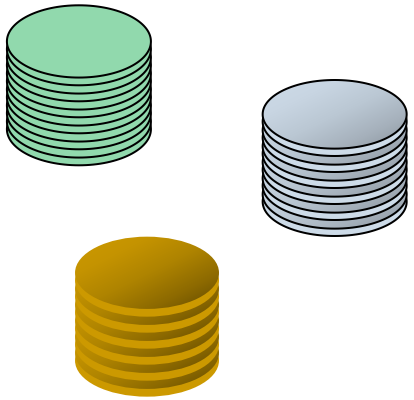
- 数据库系统概述
  - 四个基本概念
  - 数据管理的发展过程

- 数据库系统的结构
  - 数据库系统三级模式结构
  - 数据库系统的体系结构

❖ 课堂作业：

1. 数据库系统概述
   - 四个基本概念及其关系
   - 数据管理技术的发展经历了哪几个阶段

2. 数据库系统的结构
   - 数据库系统三级模式和二级映像分别指什么？画出数据库系统的体系结构图



数据库系统

第二章 MySQL的安装和配置

北京工业大学耿丹学院

计算机科学与技术专业

主要内容

- SQL概述
- MYSQL的下载与安装
- MySQL的目录结构
- MySQL服务的启动及停止
- 登录MySQL服务端
- 数据库基本操作

一、 SQL概述

- 结构化查询语言SQL（Structured Query Language）是一种介于关系代数与关系演算之间的语言，其功能包括查询、操纵、定义和控制四个方面，是一个通用的、功能极强的关系数据库语言。

1. 综合统一
2. 高度非过程化
3. 面向集合的操作方式
4. 语言简洁，易学易用

…

表 2.1　SQL 语言的动词

SQL 功　能	动　　词
数 据 定 义	CREATE，DROP，ALTER
数 据 查 询	SELECT
数 据 操 纵	INSERT，UPDATE DELETE
数 据 控 制	GRANT，REVOKE

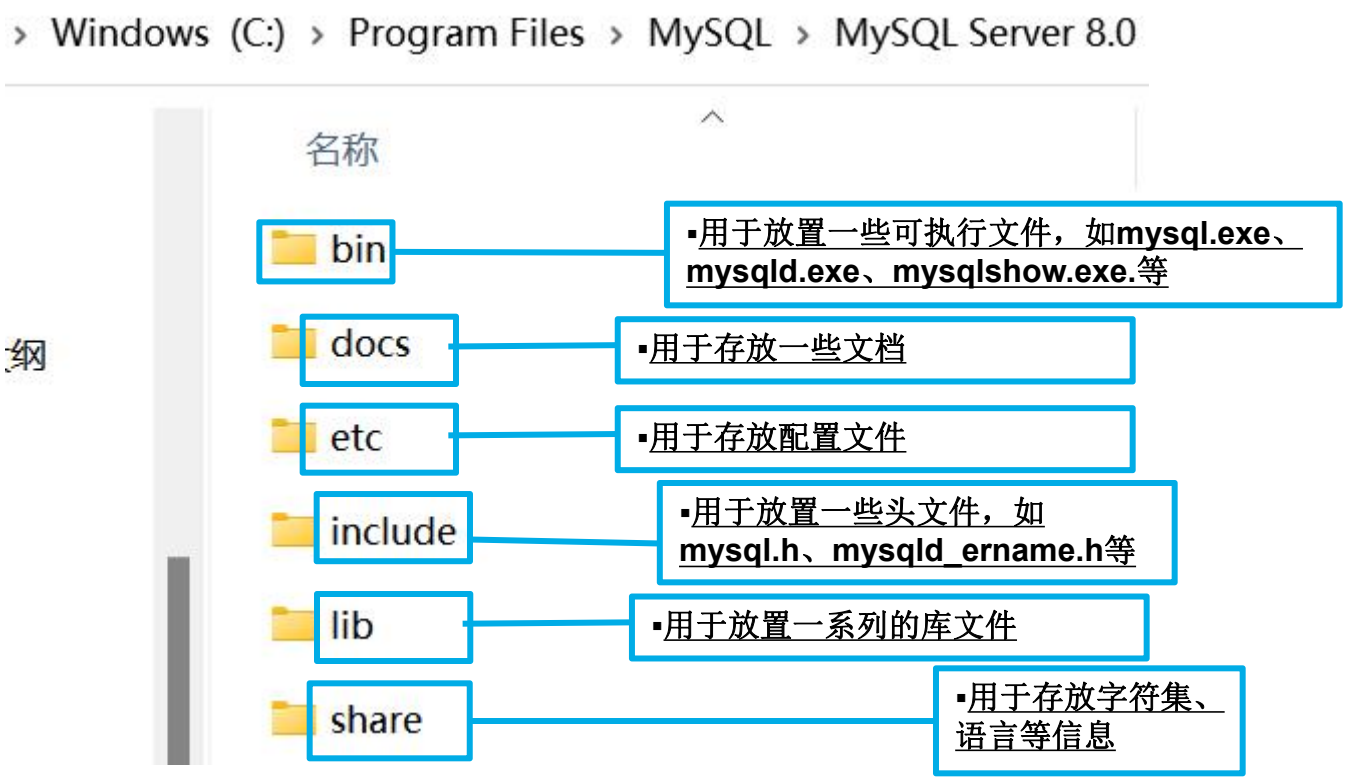
二、 MYSQL下载与安装

MYSQL8.0安装说明

Navicat 安装说明

三、 MySQL目录结构

MySQL安装完成后，会在磁盘上生成一个目录，该目录被称为MySQL的安装目录。在MySQL的安装目录中包含了启动文件、配置文件、数据库文件和命令文件等。



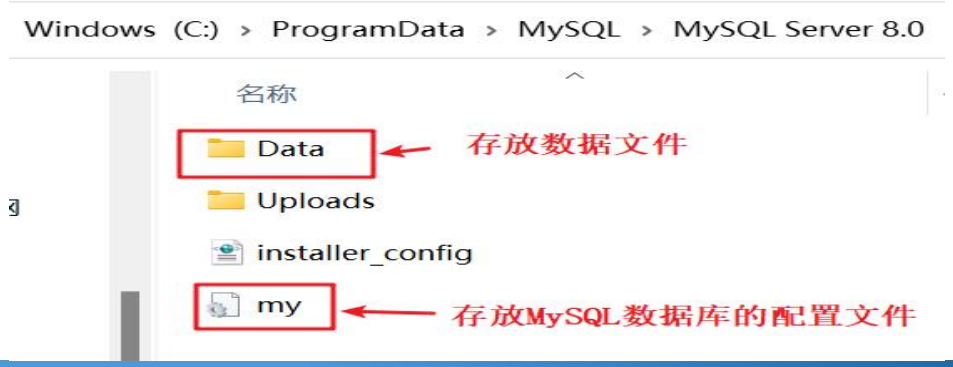
三、 MySQL目录结构

数据库文件的存放路径：

MySQL服务器程序在启动时会到文件系统的某个目录下加载一些文件，之后在运行过程中产生的数据也都会存储到这个目录下的某些文件中，这个目录就称为 数据目录。

show variables like 'datadir';

```
mysql> show variables like 'datadir';
+---------------+-----------------------------------------+
| Variable_name | Value                                   |
+---------------+-----------------------------------------+
| datadir       | C:\ProgramData\MySQL\MySQL Server 8.0\Data\ |
+---------------+-----------------------------------------+
1 row in set, 1 warning (0.01 sec)
```

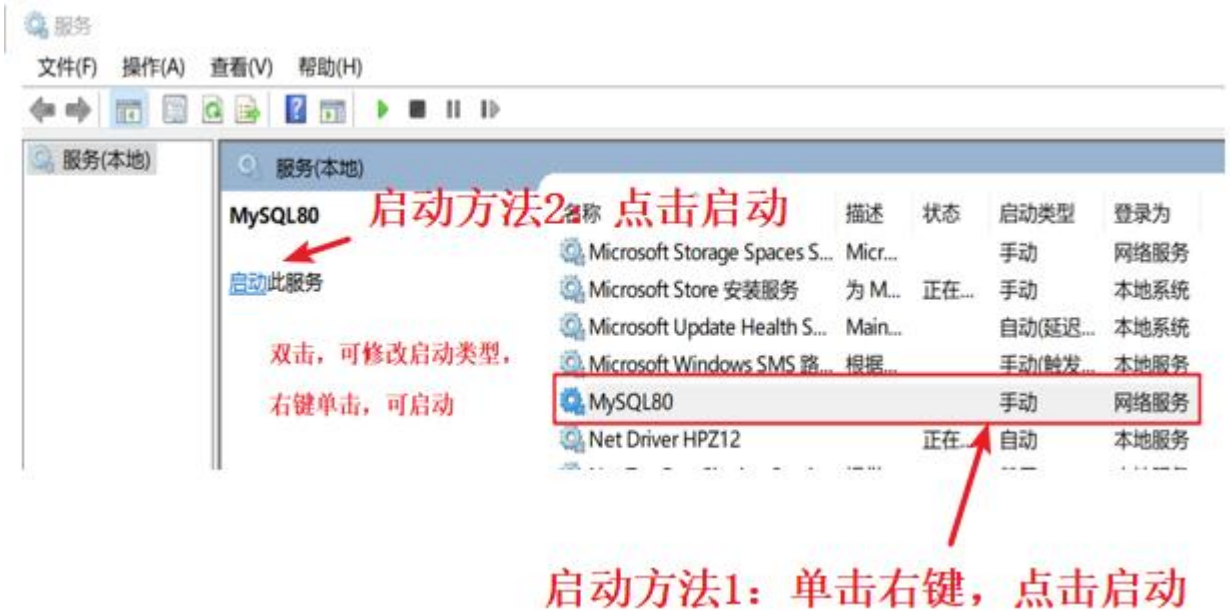


四、 MySQL服务的启动与停止

▪Windows服务管理器启动服务

▪首先开启Windows的服务管理器界面。

如果MySQL服务没有启动，可以直接双击MySQL服务项打开属性对话框



▪自动：会随系统一起启动。

▪手动：服务不会随系统一起启动，直到需要时才会被激活。

▪已禁用：服务将不再启动，即使是在需要它时，也不会被启动，除非修改为上面两种类型。

四、 MySQL服务的启动与停止

▪利用DOS命令启动服务

▪1）启动MySQL服务的具体命令为：

▪ net start mysql80(表示mysql服务器名)

▪2）停止MySQL服务的具体命令为：

▪ net stop mysql80

▪操作步骤

▪1）打开运行对话框：win+R

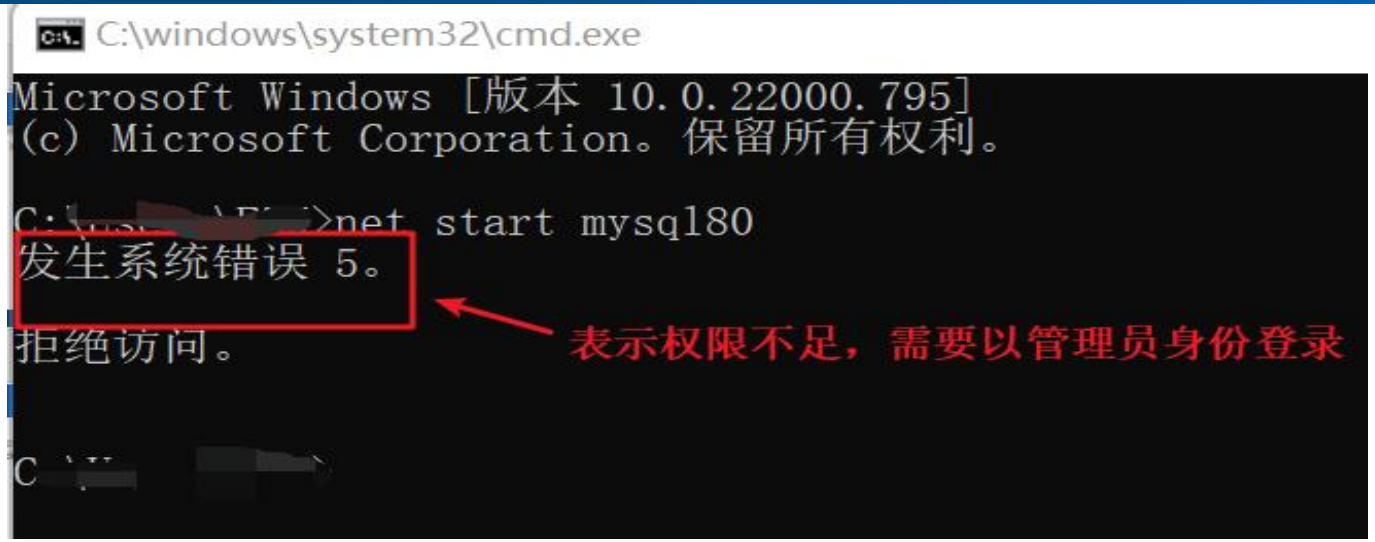
▪2) 输入cmd,打开cmd命令窗口

▪ net start mysql80

▪3）停止MySQL服务的具体命令为：

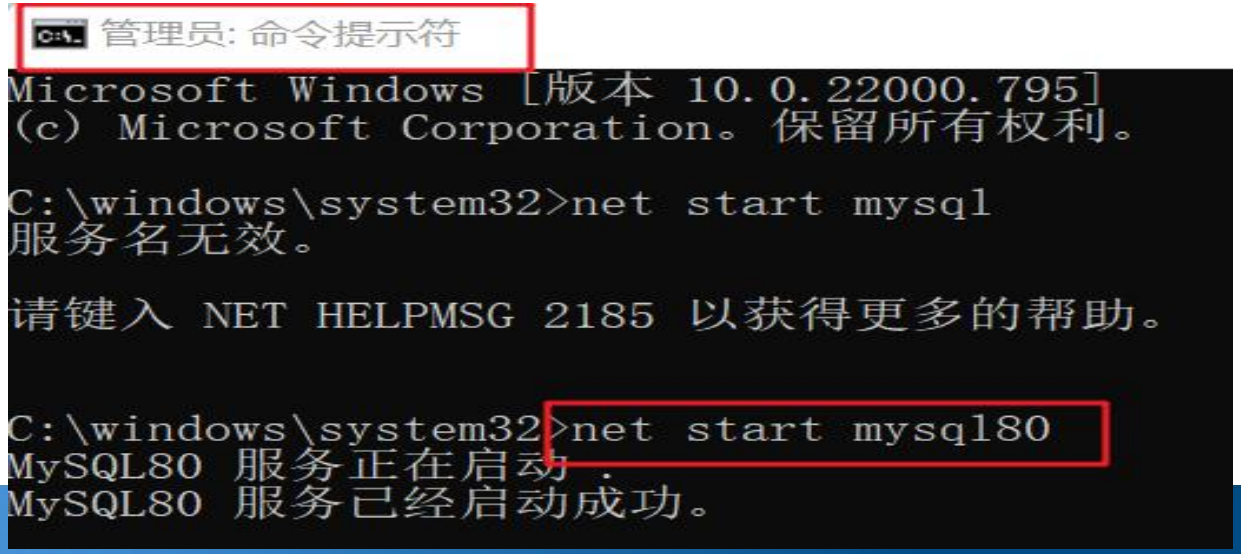
▪ net stop mysql80

常见问题及处理方法



出现这样的问题主要是是因为当前用户的操作权限太低了

解决方法：



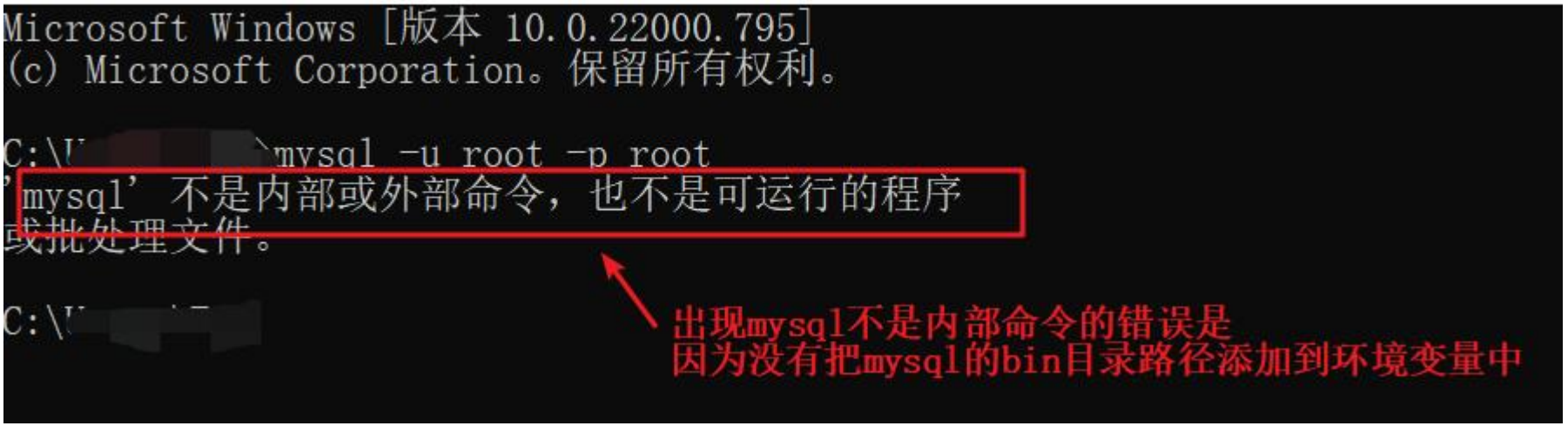
五、登录MySQL服务器

- 通过命令方式
- 通过MySQL自带的命令行客户端
- 通过MySQL组件Workbench客户端
- 通过第三方客户端SQLyog/Navicat等

五、登录MySQL服务器

方法一：通过命令方式（需要先进入DOS窗口）

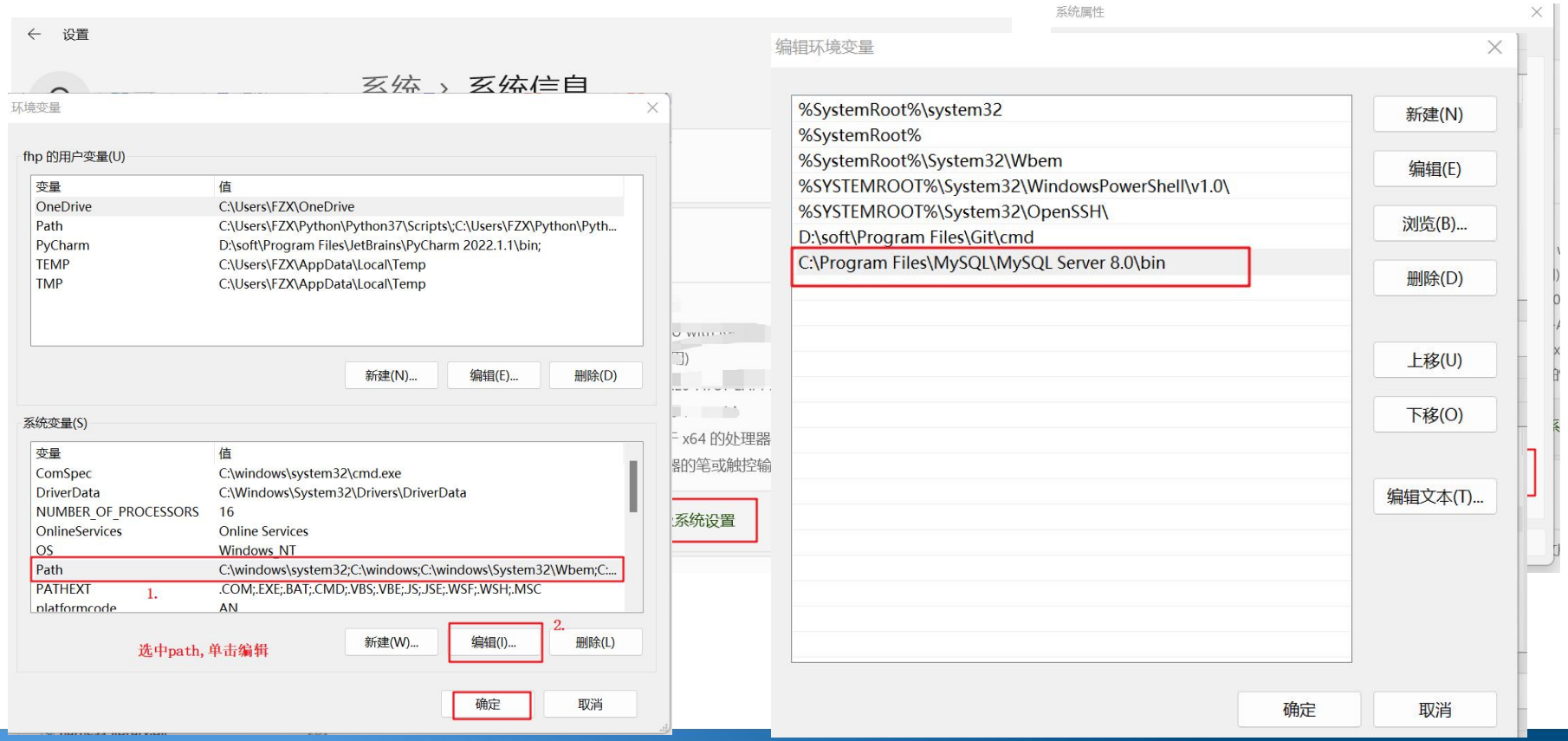
- **mysql –h hostname –u username –p password**
- 在上述命令中，mysql为登录命令，-h后面的参数是服务器的主机地址（localhost 或者127.0.0.1，可省略），-u后面的参数是登录数据库的用户名，-p后面是登录密码



方法一：通过命令方式登录服务器常见问题及处理方法

出现MySQL不是内部命令的解决方案：添加环境变量

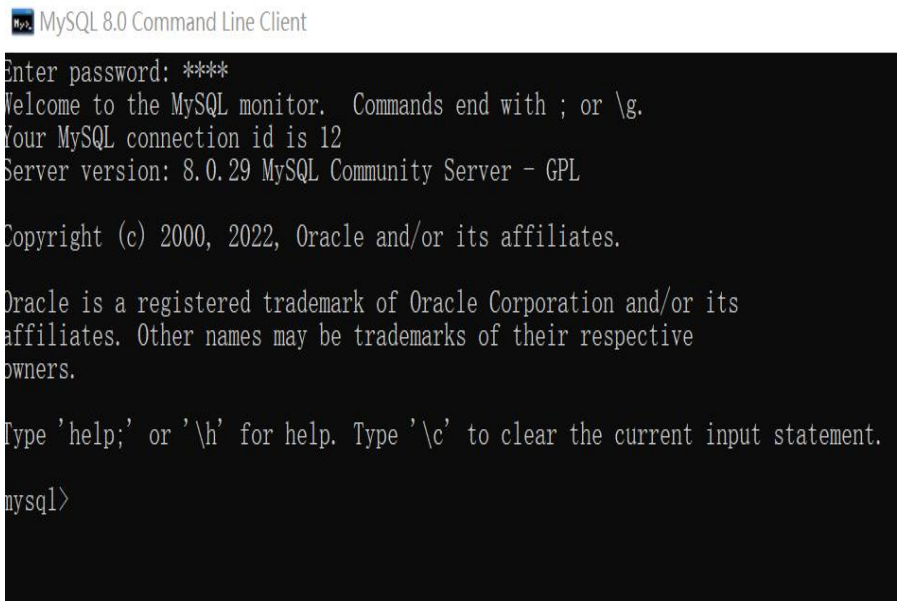
以win 11系统为例，添加环境变量的基本步骤：
搜索中输入系统--->高级系统设置-->环境变量→Path→编辑-->新建→确定

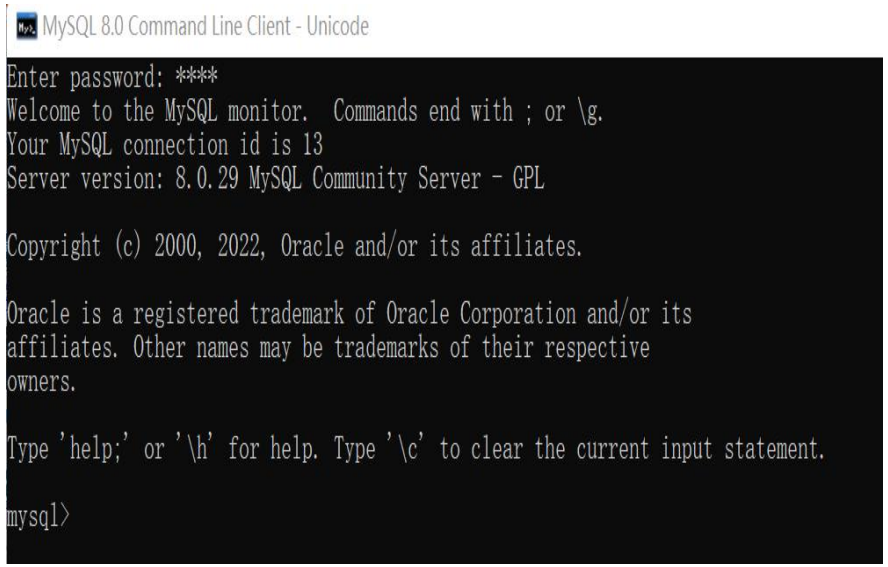


五、登录MySQL服务器

方法二：通过MySQL自带的命令行客户端

▪在开始菜单中依次选择【程序】→【MySQL】→【MySQL 8.0 Command Line Client/Unicode】打开MySQL命令行客户端窗口，此时就会提示输入密码，密码输入正确后便可以登录到MySQL数据库

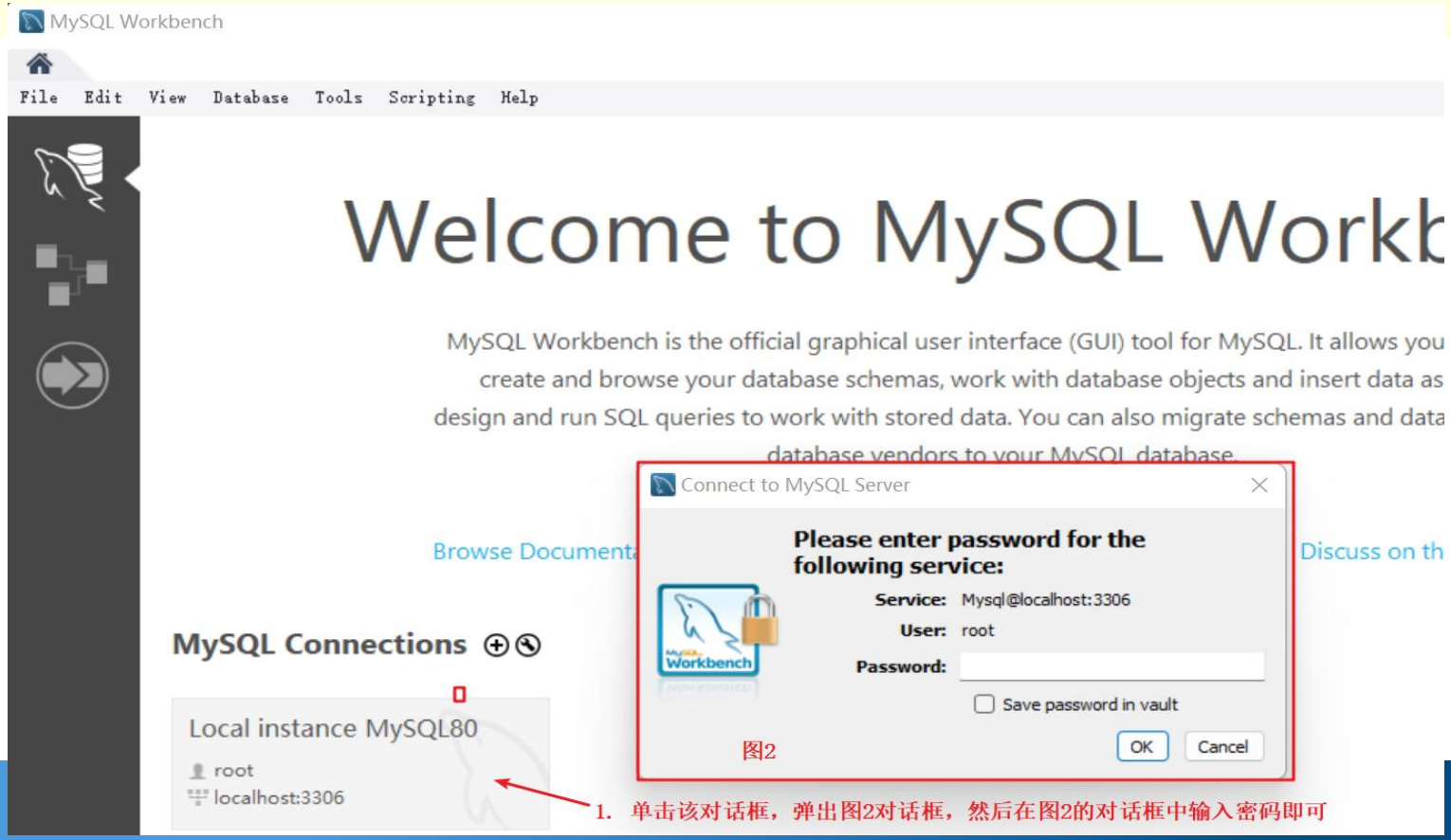




五、登录MySQL服务器

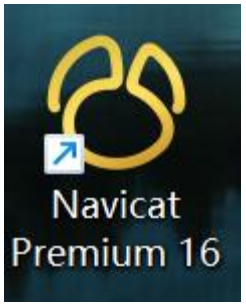
方法三：通过MySQL的组件Workbench

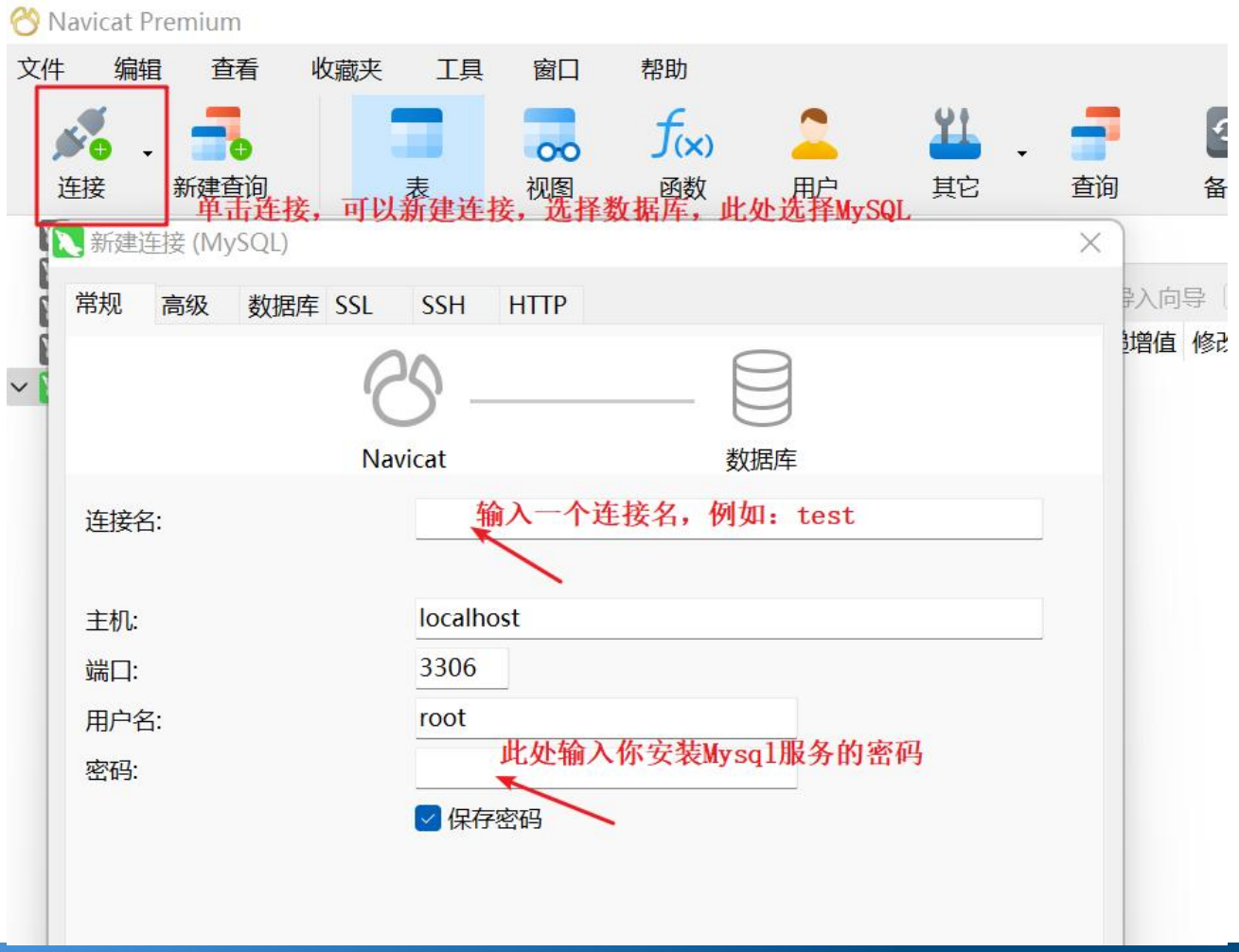
- 在开始菜单中依次选择【程序】→【MySQL】→【MySQL 8.0 workbench CE】打开MySQL命令行客户端窗口，此时就会提示输入密码，密码输入正确后便可以登录到MySQL数据库



五、登录MySQL服务器

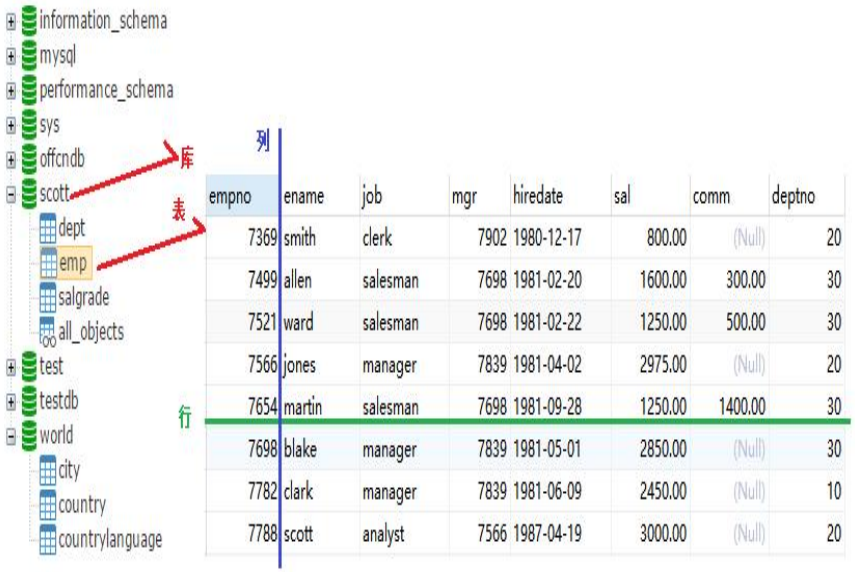
方法四：通过第三方客户端SQLyog/Navicat等





六、 数据库基本结构

- ✓ 核心存储单元是表，数据放在表中（类似于excel表）
- ✓ 表要放在数据库（相当于目录）下
- ✓ 表由列和行组成
- ✓ 列(column)也被称为字段(field)
- ✓ 行(row)也被称为记录(record)或数据(data)
- ✓ 一个表至少有一列
- ✓ 一个表可以有0到N行

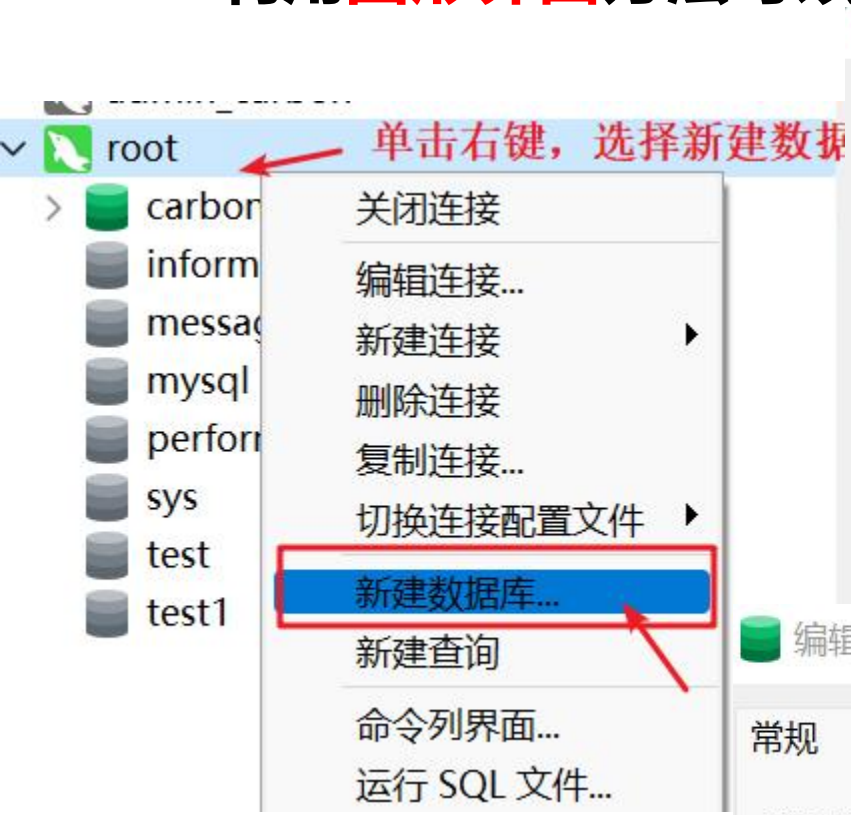


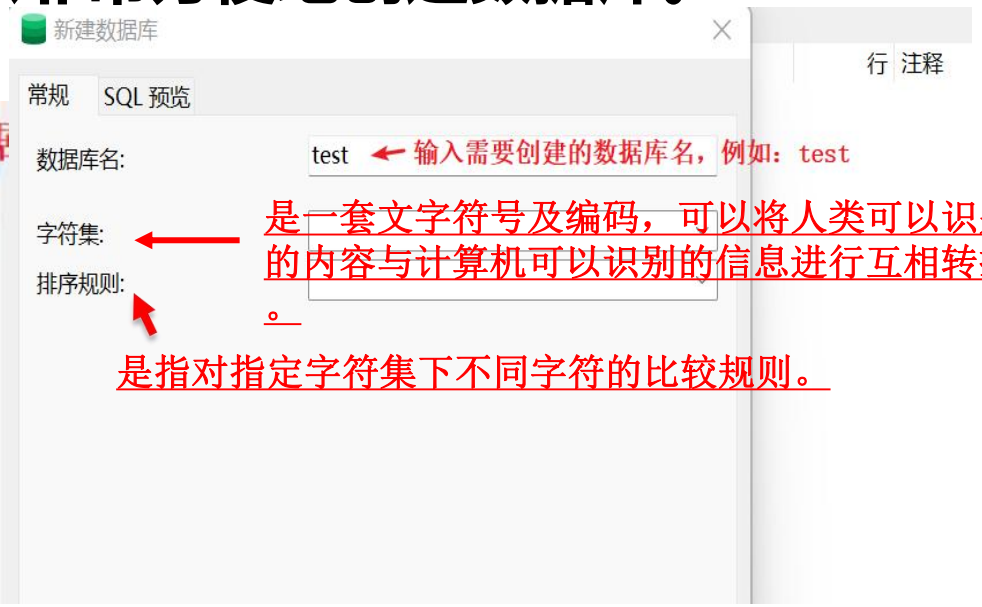
七、 数据库基本操作

- 数据库基本操作主要包括两种方法：
- 一种方法：使用图形界面方式完成对数据库的操作；
- 一种方法：使用SQL语句，以命令方式完成对数据库的操作。

1.创建数据库

- 1．利用图形界面方法可以非常方便地创建数据库。





2．利用SQL语句对数据库进行操作

（1）创建数据库

```
create database database_name;
```

（2）显示已经存在的数据库

```
show databases ;
```

（3）查看创建好的数据库的信息

```
show create database database_name ;
```

（4）修改数据库的编码

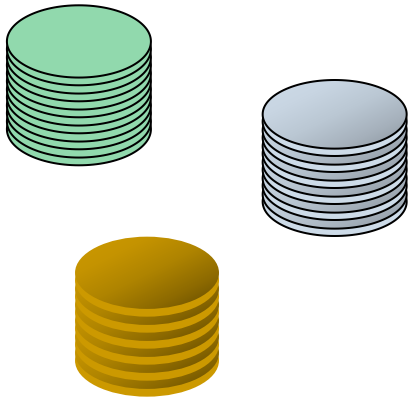
```
alter database database_name
character set 编码方式
collate 编码方式_bin;
```

（5）删除数据库

```
drop database database_name;
```

MySQL 中的注释语句

```
-- 查看数据库;
show databases;
#创建数据库
create database stu_2;
#查看stu_2数据库的基本信息
show create database stu_2;
#创建数据库stu_3,并且字符集为gbk
create database stu_3 character set gbk;
#修改数据库stu_3的字符集为utf8
alter database stu_3 character set utf8;
#查看stu_3数据库的基本信息
show create database stu_3;
#删除数据库stu_1;
drop database stu_2;
#选择数据库
use stu_3;
```



数据库系统

第三章 表的基本操作

北京工业大学耿丹学院

计算机科学与技术专业

主要内容

- 主要内容

数据表

创建表

更改表

插入数据

更新数据

- 学习目标

熟练掌握数据表的创建、修改和删除方法；

熟练掌握表数据的插入、修改和删除方法；

熟练数据表的约束及其使用。

数据表的基本概念

- 数据库是保存数据的集合，其目的在于存储和返回数据。
- 数据库中包含一个或多个表。
- 表是数据库的基本构造块。同时，表是数据的集合，是用来存储数据和操作数据的逻辑结构。
- 表是由行和列所构成，行被称为记录，是组织数据的单位；列被称为字段，每一列表示记录的一个属性。

1. 数据表

表的类型：

- 永久数据表

  永久数据表在创建后一直存储在数据库文件中，直至用户删除为止。

- 临时数据表

  临时数据表用户退出或系统修复时被自动 删除。

2. 数据类型

❖使用MySQL数据库存储数据时，不同的数据类型

决定了MySQL存储数据方式的不同。

- 一个属性选用数据类型一般从两个方面考虑：

一是取值范围，二是要做哪些运算。

▪表1 Mysql数值类型

整数数据类型

数据类型	取值范围	说明	单位
TINYINT	符号值：–127 ~ 127 无符号值：0 ~ 255	最小的整数	1 字节
BIT	符号值：–127 ~ 127 无符号值：0 ~ 255	最小的整数	1 字节
BOOL	符号值：–127 ~ 127 无符号值：0 ~ 255	最小的整数	1 字节
SMALLINT	符号值：–32768 ~ 32767 无符号值：0 ~ 65535	小型整数	2 字节
MEDIUMINT	符号值：–8388608 ~ 8388607 无符号值：0 ~ 16777215	中型整数	3 字节
INT	符号值：–2147683648 ~ 2147683647 无符号值：0 ~ 4294967295	标准整数	4 字节
BIGINT	符号值：– 9223372036854775808 ~ 9223372036854775807 无符号值：0 ~ 18446744073709551615	大整数	8 字节

浮点数据类型

数据类型	取值范围	说明	单位
FLOAT	+(–)3.402823466E+38	单精度浮点数	8 或 4 字节
DOUBLE	+(–)1.7976931348623157E+308 +(–)2.2250738585072014E-308	双精度浮点数	8 字节
DECIMAL	可变	一般整数	自定义长度

▪Mysql数值类型的选择原则

选择类型应遵循以下原则：

(1) 选择最小的可用类型，如果值永远不超过127，则使用tinyint;

(2) 对于完全都是数字的，可以选择整数类型。

(3) 浮点类型用于可能具有小数部分的数。例如货物金额等

字符串类型

字符串类型：

(1) 普通字符串：char,varchar

(2)可变类型：text(适合存储长文本),blob(适合存储二进制文件)

(3) 特殊类型： set(集合类型),enum（枚举类型）

表2 字符串类型

类　　型	取值范围	说　　明
[national] char(M) [binary|ASCII|unicode]	0～255 个字符	固定长度为 M 的字符串，其中 M 的取值范围为 0～255。National 关键字指定了应该使用的默认字符集。Binary 关键字指定了数据是否区分大小写（默认是区分大小写的）。ASCII 关键字指定了在该列中使用 latin1 字符集。Unicode 关键字指定了使用 UCS 字符集
char	0～255 个字符	Char(M) 类似
[national] varchar(M) [binary]	0～255 个字符	长度可变，其他和 char(M) 类似

类　　型	最大长度（字节数）	说　　明
TINYBLOB	2^8～1(225)	小 BLOB 字段
TINYTEXT	2^8～1(225)	小 TEXT 字段
BLOB	2^16～1(65535)	常规 BLOB 字段
TEXT	2^16～1(65535)	常规 TEXT 字段
MEDIUMBLOB	2^24～1(16777215)	中型 BLOB 字段
MEDIUMTEXT	2^24～1(16777215)	中型 TEXT 字段
LONGBLOB	2^32～1(4294967295)	长 BLOB 字段
LONGTEXT	2^32～1(4 294967295)	长 TEXT 字段

表3 枚举和集合类型

数 据 类 型	注　释
enum类型	枚举类型：enum(‘值1’, ‘值2’,……‘值n’)， 只能取其中之一
set类型	集合类型：set(‘值1’, ‘值2’,……‘值n’) 值可以取零个或多个

▪Mysql字符串类型的选择原则

字符串类型应遵循以下原则：

(1) 从速度方面考虑，要选择固定列，可使用char类型。

(2) 要节省空间，使用动态列，可以使用varchar类型。

(3) 如果要将列中的内容限制在一种选择，可以使用enum类型

(4)允许在一个列中有多余一个的条目，可使用set类型

(5)对于长文本，如果搜索的内容不区分大小写，可以使用text类型

(6)对于长文本，如果搜索的内容区分大小写，可以使用blob类型

表4 时间和日期类型

类型	大小(字节)	范围	格式	用途
DATE	3	1000-01-01/9999-12-31	YYYY-MM-DD	日期值
TIME	3	'-838:59:59'/'838:59:59'	HH:MM:SS	时间值或持续时间
YEAR	1	1901/2155	YYYY	年份值
DATETIME	8	1000-01-01 00:00:00/9999-12-31 23:59:59	YYYY-MM-DD HH:MM:SS	混合日期和时间值
TIMESTAMP	4	1970-01-01 00:00:00/2038 结束时间是第 **2147483647** 秒，北京时间 **2038-1-19 11:14:07**，格林尼治时间 2038年1月19日 凌晨 03:14:07	YYYYMMDD HHMMSS	混合日期和时间值，时间戳

Timestamp类型字段有自动更新特性

CURRENT_TIMESTAMP

Timestamp类型字段的CURRENT_TIMESTAMP

1.TIMESTAMP DEFAULT CURRENT_TIMESTAMP

ON UPDATE CURRENT_TIMESTAMP

在创建新记录和修改现有记录的时候都对这个数据列刷新

2.TIMESTAMP DEFAULT CURRENT_TIMESTAMP

在创建新记录的时候把这个字段设置为当前时间，但以后修改时，不再刷新它

3.TIMESTAMP ON UPDATE CURRENT_TIMESTAMP

在创建新记录的时候把这个字段设置为0，以后修改时刷新它

4.TIMESTAMP DEFAULT 'yyyy-mm-

dd hh:mm:ss' ON UPDATE CURRENT_TIMESTAMP

在创建新记录的时候把这个字段设置为给定值，以后修改时刷新它

```sql
-- timestamp
create table test_time(
id int,
t1 timestamp default current_timestamp
    on update current_timestamp comment '创建和更新时刷新时间',
t2 timestamp default current_timestamp comment '创建刷新时间，更新时不刷新',
t3 timestamp on update current_timestamp comment '更新时刷新时间',
t4 timestamp default '2018-01-01'
on update current_timestamp comment '创建时默认为2018-01-01，更新时刷新时间'
)comment '时间类型';
```

1 信息 | 2 表数据 | 3 信息

id	t1	t2	t3	t4
1	2018-02-28 20:51:55	2018-02-28 20:51:5	0000-00-00 00:00:00	2018-01-01 00:00:00
3	2018-02-28 20:52:15	2018-02-28 20:52:0	2018-02-28 20:52:15	2018-02-28 20:52:15
3	2018-02-28 20:52:38	2018-02-28 20:52:3	0000-00-00 00:00:00	2018-01-01 00:00:00
4	2018-02-28 20:52:47	2018-02-28 20:52:4	0000-00-00 00:00:00	2018-01-01 00:00:00
* (NULL)	CURRENT_TIMESTAMP	CURRENT_TIMESTAMP	0000-00-00 00:00:00	2018-01-01 00:00:00

3. 表的约束

完整性约束

完整性规则是对关系（表）的某种约束条件。

关系模型中三类完整性约束：

实体完整性

参照完整性

用户定义的完整性

实体完整性和参照完整性是关系模型必须满足的完整性约束条件，被称作是关系的两个不变性，应该由关系系统自动支持。

用户定义的完整性是应用领域需要遵循的约束条件。

3. 表的约束

（1）实体完整性和主码

- 实体完整性规则：要求主码不能为空，且不能取相同的值。
- 主码的定义：
- **Create table** 语句中使用 **primary Key**

a）在属性定义后加上关键字 **primary Key**；

b）在属性表定义后加上额外的定义主码的子句：

primary Key（<主码属性名表>）

3. 表的约束

（1）实体完整性和主码

说明：

1 码若由两个或以上的属性组成，则只能用上述第二种方法定义。

2 对于候选码的说明方法，可以用Unique说明该属性的值不能重复出现。Unique的使用与Primary Key相似。

3 一个表中只能有一个主码定义，但可以有多个Unique说明。

3. 表的约束

（2）参照完整性和外码

1. 关系间的引用

在关系模型中实体及实体间的联系都是
存在着关系与关系间的引用。

学生

学号	姓名	性别	专业号	年龄
801	张三	女	01	19
802	李四	男	01	20
803	王五	男	01	20
804	赵六	女	02	20
805	钱七	男	02	19

例1 学生实体、专业实体以及专业与学生间的一对多联系。

学生（学号，姓名，性别，专业号，年龄）

专业（专业号，专业名）

专业

专业号	专业名
01	信息
02	数学
03	计算机

学生关系中每个元组的“专业号”属性只取下面两类值：

（1）空值，表示尚未给该学生分配专业

（2）非空值，这时该值必须是专业关系中某个元组的“专业号”值，表示该学生不可能分配到一个不存在的专业中。

3. 表的约束

（2）参照完整性和外码

- 参照完整性是对关系间引用数据的一种限制。即：若属性组A是基本关系R1的外码，它与基本关系R2的主码K相对应，则R1中每个元组在A上的值必须：取空值，或等于R2中某元组的主码值
- SQL中就是利用外码的说明来实现参照完整性约束，限制表中某些属性的取值的。

在Create Table 语句的属性清单后，加上外码说明子句，格式为：

- FOREIGN KEY <属性名1> REFERENCES <表名>(<属性名2>)

3. 表的约束

学生（学号，姓名，性别，专业号，年龄）

课程（课程号，课程名，学分）

选修（学号，课程号，成绩）

选修（学号，课程号，成绩）

“学号”和“课程号”是选修关系中的主属性按照实体完整性和参照完整性规则，它们只能取相应被参照关系中已经存在的主码值。

学生

学号	姓名	性别	专业号	年龄
801	张三	女	01	19
802	李四	男	01	20
803	王五	男	01	20
804	赵六	女	02	20
805	钱七	男	02	19

课程

课程号	课程名	学分
01	数据库	4
02	数据结构	4
03	编译	4
04	PASCAL	2

学生选课

学号	课程号	成绩
801	04	92
801	03	78
801	02	85
802	03	82
802	04	90
803	04	88

3. 表的约束

学生（学号，姓名，性别，专业号，年龄，班长）

学生

学号	姓名	性别	专业号	年龄	班长
801	张三	女	01	19	802
802	李四	男	01	20	802
803	王五	男	01	20	802
804	赵六	女	02	20	805
805	钱七	男	02	19	805

学生（学号，姓名，性别，专业号，年龄，班长）

“班长”属性值可以取两类值：

（1）空值，表示该学生所在班级尚未选出班长，或该学生本人即是班长；

（2）非空值，这时该值必须是本关系中某个元组的学号值。

3. 表的约束

（3）用户定义完整性

- 对于用户自定义完整性约束，SQL提供了非空约束、唯一值、默认值、触发器等来实现用户的各种完整性要求。

1、非空约束：

- 在CREATE TABLE 中的属性定义后面加上NOT NULL关键字

2、唯一性约束

- 在CREATE TABLE 中的属性定义后面加上unique关键字

3、默认值约束

- 在CREATE TABLE 中的属性定义后面加上
- Default 默认值

4. 创建数据表

创建数据表的一般步骤为：

- 首先定义表结构，即给表的每一列取列名，并确定每一列的数据类型、数据长度、列数据是否可以为空等；
- 然后，为了限制某列数据的取值范围，以保证输入数据的正确性和一致性而设置约束；
- 最后就可以向表中输入数据了。

4. 创建数据表

创建数据表的方法：

- 图形化界面来操作，
- 利用SQL语句来实现。

4. 创建数据表

利用SQL语句创建数据表

创建表语法格式：

```
create  table  <表名>
（<列名> <数据类型>  [列级完整性约束条件],
 <列名> <数据类型>  [列级完整性约束条件],
  ……,
 [表级完整性约束条件]）；
```

4.创建数据表

例子：创建学生表S

```
create table s
(sno int primary key,
sname varchar(10) unique,
ssex  enum('男','女') default '男',
sage tinyint unsigned,
sdept varchar(20)
);
```

4.创建数据表

例子：创建课程表

```
create table c
(cno int  primary key,    /* 列级完整性约束条件，Cno为主码*/
cname varchar(10) not null,
cpno tinyint,
ccredit tinyint,
foreign key (cpno) references c(cno) /* 表级完整性约束条件，
Cpno为外码，被参照表是C,被参照列是Cno*/ );
```

4.创建数据表

例子：创建学生选课表

```
create table sc
( Sno int unsigned, cno int , Grade float,
 primary key(Sno,Cno),
 foreign key (Sno) references S(Sno),
 foreign key (Cno) references C (Cno));
```

注意：

后面三行均表示：表级完整性约束条件

第三行表示：主码由两个属性Sno，Cno组成

第四行表示：Sno为外码，被参照表是S；

第四行表示：Cno为外码，被参照表是C;

5.查看数据表

查看数据表有两种方式：

（1）show create table 表名。

（2）describe 表名（desc表名）

```
show create table s;
```

1 结果 | 2 信息 | 3 表数据 | 4 信息

(只读)

```
e   Create Table
--  ------------------------------------------------------------------------
    CREATE TABLE `s` (
      `sno` int(10) unsigned NOT NULL AUTO_INCREMENT COMMENT '学号',
      `sname` varchar(20) DEFAULT NULL,
      `ssex` enum('男','女') DEFAULT '男',
      `sage` int(11) DEFAULT NULL,
      `sdept` varchar(20) DEFAULT NULL,
      PRIMARY KEY (`sno`),
      UNIQUE KEY `sname` (`sname`)
    ) ENGINE=InnoDB DEFAULT CHARSET=utf8
```

```
7   desc s;
8
```

1 结果 | 2 信息 | 3 表数据 | 4 信息

(只读)

Field	Type		Null	Key	Default		Extra
sno	int(10) unsi...	16B	NO	PRI	(NULL)	0K	auto_increment
sname	varchar(20)	11B	YES	UNI	(NULL)	0K	
ssex	enum('男','女')	17B	YES		男	3B	
sage	int(11)	7B	YES		(NULL)	0K	
sdept	varchar(20)	11B	YES		(NULL)	0K	

6. 修改数据表

使用SQL语句 语法格式：

ALTER TABLE 表名称

- **rename [to]** 新表名　　#修改表名
- **add** 新字段名 数据类型 [约束条件] [first|after 已存在字段名]
- 　　first 参数表示直接添加到第一个位置，后面没有参数
- **change** 旧字段名 新字段名 新数据类型 # 修改字段名（新数据类型 不能省略）
- **modify** 字段名 数据类型　#修改字段的数据类型
- **modify** 字段名1 数据类型 **first | after** 字段名2 #修改字段的排列位置
- **drop** 字段名

例如： 设已在数据库stu_mis中创建了学生表S，现在对S表按如下要求进行修改：

①修改表名：alter table 表名称 rename [to] 新表名

eg. 将表名s修改为stu_tb

alter table s rename to stu_tb;

② 增加新的字段： alter table 表名称 add 新字段名 数据类型 [约束条件] [first|after 已存在字段名]

eg.在表S中增加一个新字段“政治面貌”。

alter table s add 政治面貌 varchar(10);

1 结果 | 2 信息 | 3 表数据 | 4 信息

（只读）

Field	Type		Null	Key	Default		Extra
sno	int(10) unsi...	16B	NO	PRI	(NULL)	0K	auto_increment
sname	varchar(20)	11B	YES	UNI	(NULL)	0K	
ssex	enum('男','女')	17B	YES		男	3B	
sage	int(11)	7B	YES		(NULL)	0K	
sdept	varchar(20)	11B	YES		(NULL)	0K	
政治面貌	varchar(10)	11B	YES		(NULL)	0K	

alter table s add 政治面貌1 varchar(10) after sname;

alter table s add 政治面貌2 varchar(10) first;

Field	Type		Null	Key	Default		Extra
政治面貌2	varchar(10)	11B	YES		(NULL)	0K	
sno	int(10) unsi...	16B	NO	PRI	(NULL)	0K	auto_increment
sname	varchar(20)	11B	YES	UNI	(NULL)	0K	
政治面貌1	varchar(10)	11B	YES		(NULL)	0K	
ssex	enum('男','女')	17B	YES		男	3B	
sage	int(11)	7B	YES		(NULL)	0K	
sdept	varchar(20)	11B	YES		(NULL)	0K	
政治面貌	varchar(10)	11B	YES		(NULL)	0K	

③删除字段： alter table 表名 drop 字段名；

eg.在表S中删除名为“政治面貌1和2”的字段。

alter table s drop 政治面貌1;

alter table s drop 政治面貌2;

④修改字段名： alter table 表名 change 旧字段名 新字段名 新数据类型

注意：新数据类型不能省略（即使和原来的数据类型一样）

eg. 将s表中政治面貌改为“s_political_status”

alter table s change 政治面貌 s_political_status varchar(10);

⑤修改字段类型： alter table 表名 modify 字段名 数据类型

eg. 修改“Sname”的字段长度由原来的20改为15;

alter table s modify sname varchar(15);

⑥修改字段的排列位置： alter table 表名 modify 字段名1 数据类型 first | after 字段名2

eg. 将sage放在sname的后面

alter table s modify sage int after sname;

7.删除数据表

删除表语法格式：

- **DROP TABLE** <表名称>
- 删除表只能够删除用户表，不能够删除系统表。删除表一旦操作完成，表中数据也一并被删除，而且是无法恢复的。

例如：

- 删除表S
- **drop table S 是否正确？**

8. 管理表数据

数据的管理包括插入、修改和删除

- insert into ……values…
- update…
- delete…

(1)表中插入数据

方法一：使用图形界面方式

方法二：使用SQL语句

格式：Insert into 表名［（属性列1 ，属性列2...）］
Values （常量1，常量2...）；

- 说明：

a. 将新元组插入指定表中。其中新元组的属性列1的值为常量1，属性列2的值为常量2，...。

b. into 子句中没有出现的属性列，新元组在这些列上将取空值。

c. into 子句中若没有指明任何属性列名，则新插入的元组必须在每个属性列上均有值。

例子：在学生表中插入数据

- insert into s(sno,sname,ssex,sage,sdept)
  values(1,'小王','男',18,'IS');
- insert into s(sname,sage,sdept)
  values('小明',19,'IS');
- insert into s(sname,sdept)
  values('小丽','CS');
- insert into s
  values(4,'王一一','女',20,'MA')

思考：

能否插入（4，‘王一一’，‘女’,20，‘MA’）?

例子：在课程表中插入如下数据

cno	cname	cpno	ccredit
1	数据库_设计	5	4
2	数学	(NULL)	2
3	信息系统	1	4
4	操作系统	6	3
5	数据结构	7	4
6	数据处理	(NULL)	2
7	C语言	6	4

例子：在课程表中插入数据

```
insert into C (Cno,Cname,Ccredit) values(1,'数据库_设计',4);

insert into C(Cno,Cname,Ccredit)
values(2,'数学 ',2), (3,'信息系统 ',4),  (4,'操作系统 ',3),
       (5,'数据结构 ',4),
       (6, '数据处理 ',2),
       (7,'C语言',4);
```

mysql可以同时添加多条记录

（2）修改表数据

修改元组值的语法格式：

Update <表名> Set <列名>=<表达式>[，<列名>=<表达式>]

[where <条件>]；

说明：表示修改指定表中满足where 子句条件的元组。

set 子句给出<表达式>的值用于取代相应的属性列值。

如果省略where子句，则表示要修改表中的所有元组。

修改数据实例

```
update c Set Cpno=5 where cname='数据库_设计';

update c Set Cpno=1 where cname='信息系统';

update c Set Cpno=6 where cname='操作系统';

update c Set Cpno=7 where cname='数据结构';

update c Set Cpno=6 where cname='C语言';
```

练习

1.在学生选课表中插入如下数据

Sno	cno	Grade
1	1	94
1	2	85
1	3	88
2	2	90
3	3	80

```
insert into SC (Sno,Cno,Grade) values(1,1,94);
insert into SC values(1,2,85);
insert into SC values(1,3,88);
insert into SC values(2,2,90);
insert into SC values(3,3,80);
```

练习

2. 将学生表中所有女生的系别信息改为“IS”

3. 将课程选课表中所有成绩均改为空值

2. update s set sdept='IS'
where ssex='女'

3. update sc
set grade=null

注意：set 语句后只能跟=null，不能用is null

（3）删除表数据

- 语句格式

  DELETE　　FROM　　<表名>　　[WHERE <条件>]；

  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
  - 缺省表示要修改表中的所有元组

例如：删除学生选课表中1号学生选课信息

```
delete from SC
where sno=1
```

删除表s性别为男的记录

```
delete from s
where ssex='男’;
```

小结

本章介绍了Mysql中数据表的相关知识，其内容主要包括数据表的基本概念、数据表的创建和管理、约束和完整性，以及如何管理表数据。数据表是一种重要的数据库对象，由行和列所构成，用于存储关系数据库中的数据。

▪创建表

- create database <库名>
- create table <表名>
- （属性1 数据类型[长度][约束]，
- 属性1 数据类型[长度][约束]，
- ……
- ）

- 常见约束
- primary key：主键
- unique：唯一性
- not null：非空
- default ：默认值
- foreign key references 外键

▪修改表结构

- ALTER TABLE 表名称
- rename [to] 新表名 #修改表名
- add 新字段名 数据类型 [约束条件] [first|after 已存在字段名]
- change 旧字段名 新字段名 新数据类型 # 修改字段名（新数据类型 不能省略）
- modify 字段名 数据类型 #修改字段的数据类型
- modify 字段名1 数据类型 first | after 字段名2 #修改字段的排列位置
- drop 字段名

▪修改表数据

插入表数据

insert [into] <表名> [<目标列>]

values (数据列表)

▪更新表数据

update <表名>

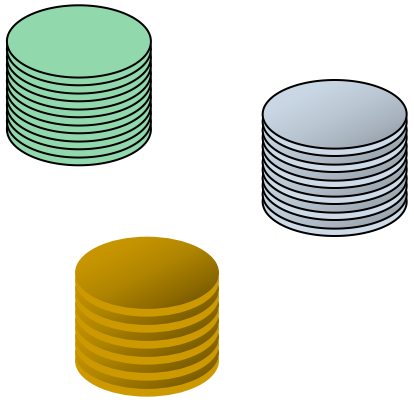
set <更新的名>=<新的表达式值>[,... n]

[where <逻辑表达式>]

▪删除表数据

delete [from] <表名>

[where <逻辑表达式>]



数据库系统

第四章 数据查询

北京工业大学耿丹学院

计算机科学与技术专业

数 据 查 询

- **单表查询**

  选择表中的若干列、选择表中的若干元组、对查询结果排序、使用集函数、对查询结果分组

- **连接查询**

  等值与非等值连接查询、自身连接、外连接、复合条件连接

- **嵌套查询**

  带有IN谓词的子查询、带有比较运算符的子查询、带有ANY或ALL谓词的子查询、带有EXISTS谓词的子查询

- **集合查询**

数据查询（单表查询）

- 无条件查询（查询表中的若干列）：

命令格式：

SELECT [ALL / DISTINCT] <目标列表达式> [,<目标列表达式>] ...
FROM <表名或视图名>;

说明：

① Select 用于描述查询输出的表结构，即输出值的列名或表达式。

② 如果在要求输出的表格中不允许有重复元组出现，可在关键词SELECT 后加DISTINCT。如果允许重复出现，加ALL（可省略）

③ 目标列表达式可以是：字段名、常量、表达式、函数，如AVG（求均值）、SUM（求和）、COUNT（*）等指定查询的内容；如果要查询所有字段，可以使用通配符“*”。

▪数据查询

基本查询

例1：查询全体学生的基本信息（显示全部列）

```
Select * From s;
```

例2：查询全体学生的学号和姓名（显示部分列）

```
Select  Sno,Sname  From s;
```

例3：查询全体学生的姓名和学号（列的顺序可以与表的不一致）

```
Select Sname ,Sno  From s;
```

例4：查询全体学生的所在系别(去除结果的重复信息)

```
Select  Sdept  From s
Select  distinct Sdept from s
```

▪数据查询

限制查询结果数量：limit offset_start,row_count
Offset_start表示数据记录的起始偏移量，默认为0
Row_count表示显示的行数；
一般limit常和order by 结合使用，即先对查询结果进行排序，然后显示部分数据记录

例：查询前2位同学的学号和姓名

```
select  sno,sname
from s
order by sno
limit 0,2;
```

或 `select sno,sname from s order by sno limit 2;`

▪数据查询

简单的计算查询（含有聚合函数的查询）用于计数和统计

除count外其他只能作用于数值型列，不考虑空值

只返回单个汇总，要想返回多个，需用分组

1. COUNT（*）——计数元组的个数

例5 求学生的总人数

```
Select count(*) From S
```

COUNT（列名）——对一列中的值计算个数

例6 找出学生所在系别的数目(列中有重复值的时候,使用distinct)

```
select COUNT(distinct(Sdept)) from S
```

2. SUM（列名）——求某一列值的总和（数值）

例7 求学生选课表中所有课程的总成绩

```
Select  sum(Grade) From SC
```

3. AVG（列名）——求某一列值的平均值（数值）

例8　求学生选课表中所有课程的平均成绩值

```
Select  Avg(Grade) From SC
```

4. Max/Min（列名）——求某一列值的最大值/最小

例9　求学生选课表中学生的最大/最小成绩值

```
Select min(Grade) From SC
```

5.求学生选课表中所有课程的平均分，最高分和最低分

```
select AVG(grade)平均分,MAX(grade)最高分,MIN(grade)最低分  from sc;
```

▪数据查询

- 有条件查询：

命令格式：

SELECT [ALL / DISTINCT] <目标列表达式> [,<目标列表达式>] ...
FROM <表名或视图名>
Where <条件表达式>

说明：

① Where 表示查询满足条件的元组
② 最常用的是比较运算符，如：=、<、>、!=或<> 、 <=，>=、!<、!>;逻辑运算符NOT、AND、OR。
③ 确定范围：between a and b（表示大于等于a,小于等于b）， not between and;
④ 确定集合：in，not in;
⑤ 字符匹配：like， not like
⑥ 空值：is null， is not null(只能用is，不能用=)

有条件查询：

例10：查询IS系全体学生的基本信息

```
Select * From S where Sdept='IS'
```

例11：查询学号为01或男同学的学生姓名

```
Select Sname From S where Sno= '01' or ssex='男'
```

例12：查询考试成绩在80分以上(含80) 的学生的学号

```
Select distinct Sno From SC where Grade>=80
```

例13：查询考试成绩在80分以上90分以下的学生的学号

```
Select distinct Sno From SC where Grade>=80
and Grade<=90;
```

确定范围：

例14：查询成绩在80-90之间的学生的学号

```
Select distinct Sno  From SC where Grade between
   80 and 90;
```

确定集合：

例15：查询IS,MA系全体学生的基本信息

```
Select  *  From S where Sdept in ('IS', 'MA')
```

或者

```
Select  *  From S where  Sdept = 'IS'  or Sdept = 'MA'
```

模糊查询

1. 语法格式：　[Not] like ‘<匹配串>’
　　[\‘<换码字符>’]。

2. 匹配符可以是一个完整的字符串，也可以含有通配符 %，_

% 代表零个或多个字符组成的字符串；

_ 代表单个字符

3. / 通配符：

在使用like进行模糊查询时，当需要搜索的字符串中包括通配符时，就需要使用/语句，将通配符转化为普通字符

模糊查询（例子）

例16：在学生表中查询姓王的学生姓名

```
Select  Sname  From S where Sname like '王%' ;
```

例17：查询名字中第二个字为“一”的学生的详细情况

```
Select  Sname  From S where Sname like '_一%';
```

例18：查询姓张或姓王的学生信息

```
select  Sname  from S
where Sname like '张%' or sname like '王%';
```

例19：查询不姓张也不姓王的学生信息

```
select  Sname  from S
where Sname not like '张%' and sname not like '王%';
```

模糊查询（例子）

/ 通配符：

在使用like进行模糊查询时，当需要搜索的字符串中包括通配符时，就需要使用/语句，将通配符转化为普通字符

例20：查询课程名为数据库_设计的课程的基本信息

```
select * from c where cname='数据库_设计';
select * from c where cname like '数据库_设计';
```

例21：查询课程名为数据库_设计的课程的基本信息

```
select * from c where cname like '数据库_设计';
select * from c where cname like '数据库/_设计' escape '/';
```

涉及空值的查询（只能用is 不能用=）

例22：查询缺少成绩的学生的学号和相应的课程号

```
Select  Sno,Cno  From SC where Grade is null;
```

例23：查询所有有成绩的学生的学号和课程号

```
Select  Sno,Cno  From SC where Grade is not null;
```

含有ORDER BY子句的查询(排序查询结果)：

命令格式：

```
SELECT [ALL / DISTINCT] <目标列表达式> [,<目标列表达式>] ...
FROM <表名或视图名>
[Where <条件表达式>]
[ORDER BY <列名2> [ASC / DESC]];
```

说明：

① Order by子句对查询结构按照一个或多个属性列的升序（ASC）或降序（DESC）排列，缺省为升序。

② 从From子句列出的表中，选取满足Where子句给出的条件表达式的元组，按Select子句中给出的列名或列名表达式求值输出。Order子句对输出的目标表进行排序，可附加说明升序或降序。

排序查询结果的例子

例24：查询学号为1的学生的所有课程的成绩，并按成绩降序排列

```
select * from sc where sno=1 order by grade desc;
```

▪数据查询

含有GROUP BY 子句的查询(分组与汇总)：

聚合函数只返回单个汇总值，而group by进行分组汇总，为结果集中的每一行产生一个汇总值

命令格式：

SELECT [ALL / DISTINCT] <目标列表达式> [,<目标列表达式>] ...
FROM <表名或视图名> [Where <条件表达式>]
[GROUP BY]<列名1> [with rollup] [HAVING <条件表达式>]]
[ORDER BY <列名2> [ASC / DESC]]；

① Group by 子句将查询结果按某一列或多列的值分组，值相等的为一组；
② All:表示对所有列和结果集（包括不满足where子句的列）分组。不满足条件的汇总列返回空值。
③ Select子句中指定的列必须是group by 子句中指定的列，或和聚合函数一起使用
④ 如果含有where子句，只对满足条件的元组分组
⑤ 分组以后如果有条件，必须使用having
⑥ WITH ROLLUP 可以实现在分组统计数据基础上再进行相同的统计（SUM,AVG,COUNT…）。

分组汇总的例子

例25：求各个课程号及相应的选课人数

```
Select Cno, count(sno) From SC group by Cno;
```

例26：统计学生选课表中每一位学生的最高分、最低分、平均分和总分

```
Select Sno,max(grade),min(grade),avg(grade),sum(grade)
From SC
group by Sno
```

例27：统计学号为1的学生的最高分、最低分、平均分和总分

```
Select Sno,max(grade),min(grade),avg(grade),sum(grade)
From SC
Where sno=1
group by Sno
```

分组汇总的例子group by … with rollup

例26’：统计学生选课表中每一位学生及所有学生选课的最高分、最低分、平均分和总分

```
Select Sno,max(grade),min(grade),avg(grade),sum(grade) From SC
group by Sno with rollup;
```

Sno	max(grade)	min(grade)	avg(grade)	sum(grade)
1	88	83	86.4000	432
2	90	90	90.0000	90
3	80	80	80.0000	80
4	70	70	70.0000	70
5	93	93	93.0000	93
(NULL)	93	70	85.0000	765

其中记录 NULL 表示所有人汇总情况。可以使用 coalesce 来设置一个可以取代 NULL 的名称，coalesce 语法：

```
select coalesce(a,b,c);
```

说明：如果a==null,则选择b；如果b==null,则选择c；如果a!=null,则选择a；如果a b c 都为null ，则返回为null（没意义）。

```
select coalesce(Sno,'总数'),max(grade),
min(grade),avg(grade),sum(grade)
from SC
group by Sno with rollup;
```

coalesce(Sno,'总数')	max(gr...	min(grade)	avg(grade)	sum(grade)
1	88	83	86.4000	432
2	90	90	90.0000	90
3	80	80	80.0000	80
4	70	70	70.0000	70
5	93	93	93.0000	93
总数	93	70	85.0000	765

分组筛选的例子

例28：统计学生选课表中每一位学生的最高分、最低分、平均分和总分，并输出平均分大于60的信息

```
Select
  Sno,max(grade),min(grade),avg(grade),sum(grade)
From SC
group by Sno
Having avg(grade)>60
```

例29：查询选修了1门以上课程的学生学号

```
Select  Sno From SC group by Sno having count(*)>1;
```

分组筛选的例子

例30：求学生表中每个系（超过1人）的总人数，要求查询结果按人数降序排列

```
Select Sdept,count(Sno) 系总人数
From S
Group by Sdept
Having 系总人数>1
Order by 2 desc
```

求学生表中每个系的总人数，要求查询结果按人数降序排列，如果人数相同，按系名降序排列

▪数据查询

一般格式　　数据查询（单表查询小结）

- SELECT [ALL / DISTINCT] <目标列表达式> [,<目标列表达式>] ...
  FROM <表名或视图名> [,<表名或视图名>] ...
  [WHERE <条件表达式>]
  [GROUP BY <列名1>[with rollup] [HAVING <条件表达式>]]
  [ORDER BY <列名2> [ASC / DESC]]
  [limit a,b];
- 说明：
- 根据WHERE子句的条件表达式，从 FROM子句指定的基本表或视图中找出满足条件的元组，再按目标列表达式表选出元组中的属性值形成结果表。
- 如果有 GROUP子句，则将结果按 <列名1> 的值进行分组，该属性列值相等的元组为一个组，每个组产生结果表中的一条记录。如果GROUP子句带with rollup短语，则再分组基础上再进行统计 。如果GROUP子句带HAVING 短语，则只有满足指定条件的组才予输出。
- 若有ORDER子句，则结果表还要按<列名2>的值的升序或降序排序。

连接查询

- 当检索数据时，往往在一个表中不能够得到想要的所有信息。通过连接操作，可以查询出存放在多个表中同一实体的不同信息，给用户带来很大的灵活性。

  多表连接实际上就是实现如何使用一个表中的数据来选择另一个表中的行。

连接的类型

- 表的连接有多种类型，当连接表时，创建的连接类型影响出现在结果集内的行。
- **1．内部连接（Inner join）**
- **2．外部连接（Outer join）**
- **3．交叉连接（cross join）**

▪数据查询

[例]

R

A	B	C
a_1	b_1	5
a_1	b_2	6
a_2	b_3	8
a_2	b_4	12

S

B	E
b_1	3
b_2	7
b_3	10
b_3	2
b_5	2

$R \times S$

A	R.B	C	S.B	E
a_1	b_1	5	b_1	3
a_1	b_1	5	b_2	7
a_1	b_1	5	b_3	10
a_1	b_1	5	b_3	2
a_1	b_1	5	b_5	2
a_1	b_2	6	b_1	3
a_1	b_2	6	b_2	7
a_1	b_2	6	b_3	10
a_1	b_2	6	b_3	2
a_1	b_2	6	b_5	2
a_2	b_3	8	b_1	3
…	…	…	…	…

等值连接 $R \bowtie S$

$R.B=S.B$

A	R.B	C	S.B	E
a_1	b_1	5	b_1	3
a_1	b_2	6	b_2	7
a_2	b_3	8	b_3	10
a_2	b_3	8	b_3	2

自然连接 $R \bowtie S$

A	B	C	E
a_1	b_1	5	3
a_1	b_2	6	7
a_2	b_3	8	10
a_2	b_3	8	2

外连接

- 左连接(Left outer join): 只保留左边关系要舍去的元组
- 右连接(Right outer join): 只保留右边关系要舍去的元组

▪数据查询

R

A	B	C
a_1	b_1	5
a_1	b_2	6
a_2	b_3	8
a_2	b_4	12

S

B	E
b_1	3
b_2	7
b_3	10
b_3	2
b_5	2

自然连接 $R \bowtie S$

A	B	C	E
a_1	b_1	5	3
a_1	b_2	6	7
a_2	b_3	8	10
a_2	b_3	8	2

A	B	C	E
a_1	b_1	5	3
a_1	b_2	6	7
a_2	b_3	8	10
a_2	b_3	8	2
a_2	b_4	12	Null
Null	b_5	Null	2

(a) R、S外连接

A	B	C	E
a_1	b_1	5	3
a_1	b_2	6	7
a_2	b_3	8	10
a_2	b_3	8	2
a_2	b_4	12	Null

(b) R、S左连接

A	B	C	E
a_1	b_1	5	3
a_1	b_2	6	7
a_2	b_3	8	10
a_2	b_3	8	2
Null	b_5	Null	2

(c) R、S右连接

❖1．内部连接（Inner join）

内部连接为典型的连接运算，使用类似于“=”或“<>”的比较运算符，它是组合两个表的常用方法。内部连接使用比较运算符根据每个表的通用列中的值匹配两个表中的行，常用的内部连接包括等值连接和自然连接。

❖2．外部连接（Outer join）

在内部连接中，只有在两个表中匹配的行才能在结果集中出现。而在外部连接中可以只限制一个表，而对另外一个表不加限制（即所有的行都出现在结果集中）。

外部连接分为左外连接、右外连接。左外连接是对连接条件中左边的表不加限制；右外连接是对右边的表不加限制；

❖3．交叉连接（Cross join）

没有WHERE子句的交叉连接将产生连接所涉及的表的笛卡尔积。第一个表的行数乘以第二个表的行数等于笛卡尔积得到的结果集的大小。

连接的实现

- 表的连接的实现可以通过两种方法：

  1.利用SELECT语句的WHERE子句。

  2.在FROM子句中使用JOIN（INNER JOIN、CROSS JOIN、OUTER JOIN、LEFT OUTER JOIN、FULL OUTER JOIN等）关键字。

❖1．使用WHERE子句实现表的连接

```
SELECT column_list
FROM table_name [ ,...n ]
WHERE { search_condition AND
join_condition }[ ,...n ]
```

其中，join_condition表示连接条件。

- 例31：查询所有选修课程号1的同学学号、姓名和成绩。

```
SELECT s.sno, Sname, grade
FROM s, sc
WHERE s.sno=sc.sno AND cno=1
```

❖2．使用JOIN关键字实现表的连接

在SELECT语句的FROM子句中，通过指定不同类型的JOIN关键字可以实现不同的表的连接方式，而在ON关键字后指定连接条件。

❖基本连接语法如下：

SELECT column_list

FROM join_table JOIN_TYPE join_table

ON (join_condition)

说明如下。

join_table：指出参与连接操作的表名。

JOIN_TYPE为连接类型，可分为3种：内部连接、外部连接和交叉连接。

❖ **1）内部连接INNER JOIN**

内部连接是使用比较运算符比较要连接列中的值的连接

❖ 【例31】查询所有选修课程号为1的同学学号、姓名和成绩。

❖ **SELECT s.sno, name, grade**

FROM s INNER JOIN sc

ON s.sno=sc.sno and cno=1

❖ 或者Where sc.cno=1

❖【例32】查询所有选修课程号为1的同学学号、姓名、成绩和课程名。

法一：

```
select s.sno,sname,cname,Grade
from s inner join sc on s.Sno=sc.Sno inner join c
    on c.cno=sc.cno
where sc.cno=1
```

法二：

```
select s.sno,sname,cname,Grade
from s, sc ,c
where s.Sno=sc.Sno  and  c.cno=sc.cno and sc.cno=1
```

❖2）外部连接OUTER JOIN

仅当两个表中都至少有一行符合连接条件时，内部连接才返回行。内部连接消除了与另一个表中的行不匹配的行。而外部连接会返回FROM子句中提到的至少一个表或视图中的所有行，只要这些行符合任何WHERE或HAVING搜索条件。将检索通过左外部连接引用的左表中的所有行，以及通过右外部连接引用的右表中的所有行；在全外部连接中，将返回两个表的所有行。

左连接——LEFT【OUTER】JOIN ON
右连接——RIGHT【OUTER】JOIN ON

[例 33] 查询每个学生及其选修课程的情况
包括没有选修课程的学生----用外连接操作

```
SELECT S.*,Cno,Grade
FROM S LEFT OUTER JOIN SC
    ON S.Sno=SC.Sno
```

❖3）交叉连接 CROSS JOIN

没有WHERE子句的交叉连接将产生连接所涉及的表的笛卡尔积。笛卡尔积结果集的大小为第一个表的行数乘以第二个表的行数。实际上交叉连接没有实际意义，通常只是用于测试所有可能的情况。

❖例34，求所有学生的所有可能的选课情况

```
select S.*,C.*   from S cross join C
```

自身连接

- 一个表与其自己进行连接，称为表的**自身连接**
- 需要给**表起别名**以示区别
- 由于所有属性名都是同名属性，因此必须使用**别名前缀**

[例35] 查询每一门课的间接先修课（即先修课的先修课）

```
select  c1.cno, c1.cname,c2.cpno
from  c C1  left outer join c c2
on  C1.Cpno=c2.cno
```

复合条件连接

WHERE子句中含多个连接条件时，称为复合条件连接

[例36]查询选修2号课程且成绩在90分以上（含90分）的所有学生的学号、姓名

```
SELECT S.Sno, S.Sname
FROM   S, SC
WHERE  S.Sno = SC.Sno      /* 连接谓词*/
       AND
       SC.Cno= 2 AND       /* 其他限定条件 */
       SC.Grade >= 90      /* 其他限定条件 */
```

多表连接

[例37] 查询选课学生的学号、姓名、选修的课程名及成绩。

```
SELECT S.Sno,Sname,Cname,Grade
FROM   S,SC,C
WHERE S.Sno = SC.Sno
      and SC.Cno = C.Cno
```

嵌套查询

- 一个SELECT-FROM-WHERE语句称为一个查询块
- 将一个查询块嵌套在另一个查询块的WHERE子句或
  - 子查询的Select语句中不能包含order by 子句。

例38：在学生表中查找选修了2号课程的学生姓名

```
SELECT Sname
 FROM S,SC
 WHERE S.Sno=SC.Sno and SC.Cno=2
```

```
SELECT Sname                    /*外层查询/父查询*/
   FROM S
   WHERE Sno IN
      （SELECT Sno              /*内层查询/子查询*/
        FROM SC
        WHERE Cno= 2）；
```

一、带有IN谓词的子查询

[例39] 查询与“李勇”在同一个系学习

此查询要求可以分步来完成

方法一：

① 确定“李勇”所在系名

```
SELECT  Sdept
FROM    S
WHERE  Sname= '李勇'
```

② 查找所有在CS系学习的学生。

```
SELECT   Sno,Sname
```

③ 将第一步查询嵌入到第二步查询的条件中。

```
SELECT Sno,Sname
FROM S
WHERE Sdept  IN
      (SELECT Sdept
       FROM S
       WHERE Sname= '李勇')
```

不相关子查询:子查询的查询条件不依赖于父查询

其求解方法是由里向外逐层处理。即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找条件。

[例39] 查询与“李勇”在同一个系学习的学生的学号和姓名。

方法二：用自身连接完成查询要求

```
SELECT  S1.Sno，S1.Sname
FROM    S S1，  S S2
WHERE  S1.Sdept = S2.Sdept  AND
            S2.Sname = ‘李勇'
```

父查询和子查询中的表均可以定义别名

```
SELECT Sno，Sname
FROM   S  S1
WHERE  S1.Sdept  IN
      (SELECT Sdept
       FROM    S S2
       WHERE S2.Sname= ‘ 李勇 ’ );
```

[例40]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S关系中
FROM   S                      取出Sno和Sname
WHERE Sno  IN
      (SELECT Sno             ② 然后在SC关系中找出选
       FROM
       WHERE
        (SELE
        FROM
        WHE
```

用连接查询

```
SELECT Sno，Sname
FROM   S，SC，C
WHERE S.Sno = SC.Sno  AND
            SC.Cno = C.Cno AND
            C.Cname=‘信息系统’；
```

二、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，可用比较运算符（>，<，=，>=，<=，!=或< >）。

例：假设一个学生只可能在一个系学习，并且必须属于一个系，则在[例39]可以用 = 代替IN：

```
SELECT Sno，Sname，S
FROM   S
WHERE Sdept  =
      （SELECT Sdept
        FROM   S
        WHERE Sname= ‘
```

子查询一定要跟在比较符之后

错误的例子：

```
SELECT  Sno，Sname，Sdept
FROM    S
WHERE ( SELECT Sdept
  FROM S
    WHERE Sname= ‘刘晨’ )
        = Sdept;
```

[例41]找出每个学生超过他选修课程平均成绩的课程号。

```
SELECT  Sno，Cno
FROM    SC  x
WHERE Grade>=( SELECT AVG(Grade)
                      FROM SC y
                      WHERE y.Sno=x.Sno );
```

相关子查询

- 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。

三、带有ANY或ALL谓词的子查询

子查询返回单值时可以比较运算符，但返回多值时要用ANY或ALL谓词修饰符,需要配合使用比较运算符

> 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

[例42] 查询其他系中比计算机科学系某一学生年龄小的学生姓名和年龄

```
SELECT Sname,Sage
    FROM    S
    WHERE Sage  < ANY (SELECT  Sage
                FROM    S
                WHERE Sdept= 'CS')
          AND Sdept <> 'CS' ;          /* 注意这是父查询块中的条件*/
```

执行过程:

1.DBMS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(19，20)

2. 处理父查询，找所有不是CS系且年龄小于19 或 20的学生

[例42 ‘]：用聚合函数实现[例42]

```
SELECT Sname,Sage
FROM S
WHERE Sage <
        (SELECT max(Sage)
         FROM S
         WHERE Sdept= 'CS')
      AND Sdept <> 'CS';
```

执行过程:

1.DBMS执行此查询时，首先处理子查询，求出CS系中所有学生的年龄最大的值(20)

2. 处理父查询，找所有不是CS系且年龄小于20的学生

用集函数实现子查询通常比直接用ANY或ALL查询效率要高，因为前者通常能够减少比较次数

ANY和ALL谓词有时可以用聚合函数实现

ANY与ALL与聚合函数的对应关系

-	=	<>或!=	<	<=	>	>=
ANY	IN	--	<MAX	<=MAX	>MIN	>= MIN
ALL	--	NOT IN	<MIN	<= MIN	>MAX	>= MAX

[例43] 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。

方法一：用ALL谓词

```
SELECT Sname,Sage
FROM S
WHERE Sage < ALL
      (SELECT Sage
       FROM S
       WHERE Sdept= 'CS')
  AND Sdept <> 'CS';
```

方法二：用聚合函数

```
SELECT Sname,Sage
FROM S
WHERE Sage <
      (SELECT MIN(Sage)
       FROM S
       WHERE Sdept= 'CS')
  AND Sdept <>'CS';
```

- 子查询除了可用在select语句中，还可用在insert、update、delete语句中。

插入数据

一、插入单个元组

- 语句格式

  INSERT INTO <表名> [(<属性列1>[，<属性列2 >…)]
  VALUES (<常量1> [，<常量2>] …)

- 功能：将新元组插入指定表中。

[例46] 将一个新学生记录:（学号：10；姓名：陈冬；性别：男；年龄：18岁；所在系：IS；）插入到S表中。

```
INSERT INTO S
VALUES (' 10 '，'陈冬'，'男'，18，'IS');
```

[例47] 插入一条选课记录(‘10'，'1 ')。

```
INSERT INTO SC(Sno，Cno)
VALUES (' 10 '，' 1 ');
```

新插入的记录在Grade列上取空值

二、插入子查询结果

- 语句格式

  INSERT INTO <表名> [(<属性列1> [，<属性列2>…)] 子查询；

- 功能：将子查询结果插入指定表中

[例48] 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 Deptage
 (Sdept CHAR(15),        /* 系名*/
 Avgage INT) /*学生平均年龄*/
```

第二步：插入数据

```
INSERT INTO Deptage(Sdept,Avgage)
    SELECT Sdept,AVG(Sage)
    FROM S
    GROUP BY Sdept
```

修改数据

- 语句格式

  UPDATE <表名>

  SET <列名>=<表达式>[，<列名>=<表达式>]…

  [WHERE <条件>]；

- 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、 修改后取值。
- **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

▪数据查询

[例49] 将学生1的年龄改为22岁。

```
UPDATE  S
 SET Sage=22
WHERE  Sno=1;
```

[例50] 将所有学生的年龄增加1岁。

```
UPDATE S
SET Sage= Sage+1;
```

[例50`] 将信息系所有学生的年龄增加1岁。

```
UPDATE S
SET Sage= Sage+1
WHERE Sdept=' IS ';
```

[例51] 将计算机科学系全体学生的成绩置零。

```
UPDATE SC
SET  Grade=0
WHERE  Sno in
      (SELETE Sno
       FROM  S
       WHERE  Sdept='CS');
```

删除数据

- 语句格式

  DELETE　　FROM　　<表名>　　[WHERE <条件>];

  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
  - 缺省表示要修改表中的所有元组

- 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

[例52] 删除学号为4的学生记录。

```
DELETE
    FROM S
    WHERE Sno=4;
```

[例53] 删除所有的学生选课记录。

```
DELETE
    FROM SC;
```

[例54] 删除2号课程的所有选课记录。

```
DELETE
    FROM SC;
    WHERE Cno=2;
```

[例55] 删除计算机科学系所有学生的选课记录。

```
DELETE
    FROM  SC
    WHERE  sno  in
      (SELETE Sno
       FROM  S
       WHERE  Sdept='CS';
```

集合查询

标准SQL直接支持的集合操作种类

- 并操作(UNION)

一般商用数据库支持的集合操作种类

- 并操作(UNION)
- 交操作(INTERSECT)
- 差操作(Except/Minus)

MySQL只支持并操作。不支持交和差

1. 并操作

形式　<查询块>
UNION
<查询块>

参加UNION操作的各结果表的列数必须相同；对应项的数据类型也必须相同。

[例55] 查询计算机科学系的学生或年龄不大于20岁的学生。

方法一：

```
SELECT *
FROM S
WHERE Sdept= 'CS'
UNION
SELECT *
FROM S
WHERE Sage<=20;
```

方法二：

```
SELECT DISTINCT *
FROM S
WHERE Sdept= 'CS' OR
Sage<=20;
```

[例56] 查询选修了课程1或者选修了课程2的学生。

方法一：

```
SELECT Sno
FROM SC
WHERE Cno=1
UNION
SELECT Sno
FROM SC
WHERE Cno= 2 ;
```

方法二：

```
SELECT DISTINCT Sno
FROM SC
WHERE Cno=1 OR Cno= 2 ;
```

2．交操作

[例57] 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 *
 FROM S
 WHERE Sdept= ‘CS’
INTERSECT
 SELECT *
  FROM S
  WHERE   Sage<=19
```

```
SELECT *
    FROM S
    WHERE Sdept= 'CS'
       AND  Sage<=19;
```

本例实际上就是查询计算机科学系中年龄不大于19岁的学生。

[例58] 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
 FROM SC
 WHERE Cno=1
INTERSECT
 SELECT Sno
 FROM SC
 WHERE Cno=2
```

```
SELECT Sno
  FROM SC
  WHERE Cno=' 1 ' AND Sno IN
              (SELECT Sno
                FROM SC
                WHERE Cno=2);
```

本例实际上是查询既选修了课程1又选修了课程2的学生。

3．差操作

标准SQL中没有提供集合差操作，但可用其他方法间接实现。

[例59] 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 *
 FROM S
 WHERE Sdept= ‘CS’
EXCEPT
 SELECT *
  FROM S
  WHERE   Sage<=19
```

```
SELECT *
  FROM S
   WHERE Sdept= 'CS' AND
          Sage>19;
```

本例实际上是查询计算机科学系中年龄大于19岁的学生。

查询总结

SELECT语句的一般格式

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

1、目标列表达式

(1) *

(2) [<表名>.] *

(4) COUNT ([DISTINCT|ALL] *)

(4)[<表名>.]<属性列名表达式>[，[<表名>.]<属性列名表达式>] …

<属性列名表达式>：由属性列、作用于属性列的聚合函数和常量的任意算术运算（+，-，*，/）组成的运算公式。

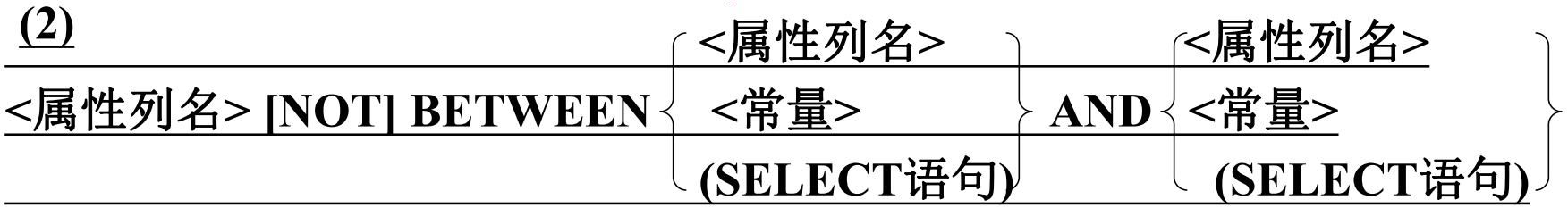
2、聚合函数格式

COUNT
SUM
AVG
MAX
MIN
([DISTINCT|ALL] <列名>)

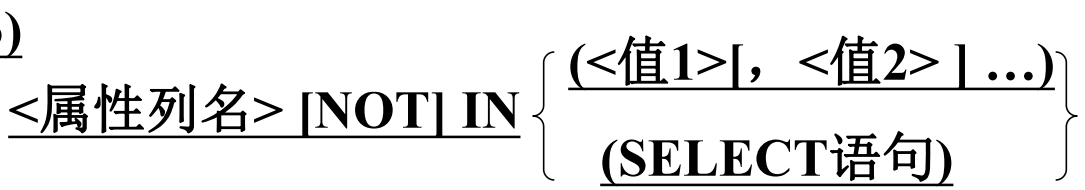
3、WHERE子句条件表达式格式

(1)

(2)



(3)

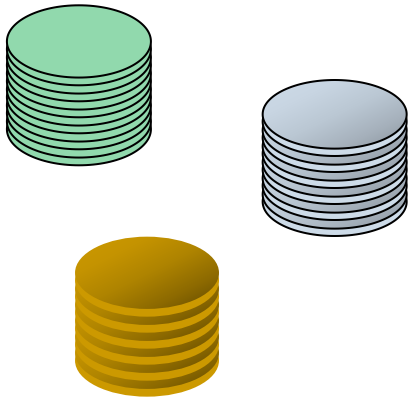


(4) **<属性列名> [NOT] LIKE <匹配串>**

(5) **<属性列名> IS [NOT] NULL**

(6)

<条件表达式> { **AND** / **OR** } **<条件表达式>** [{ **AND** / **OR** } **<条件表达>**] **…**



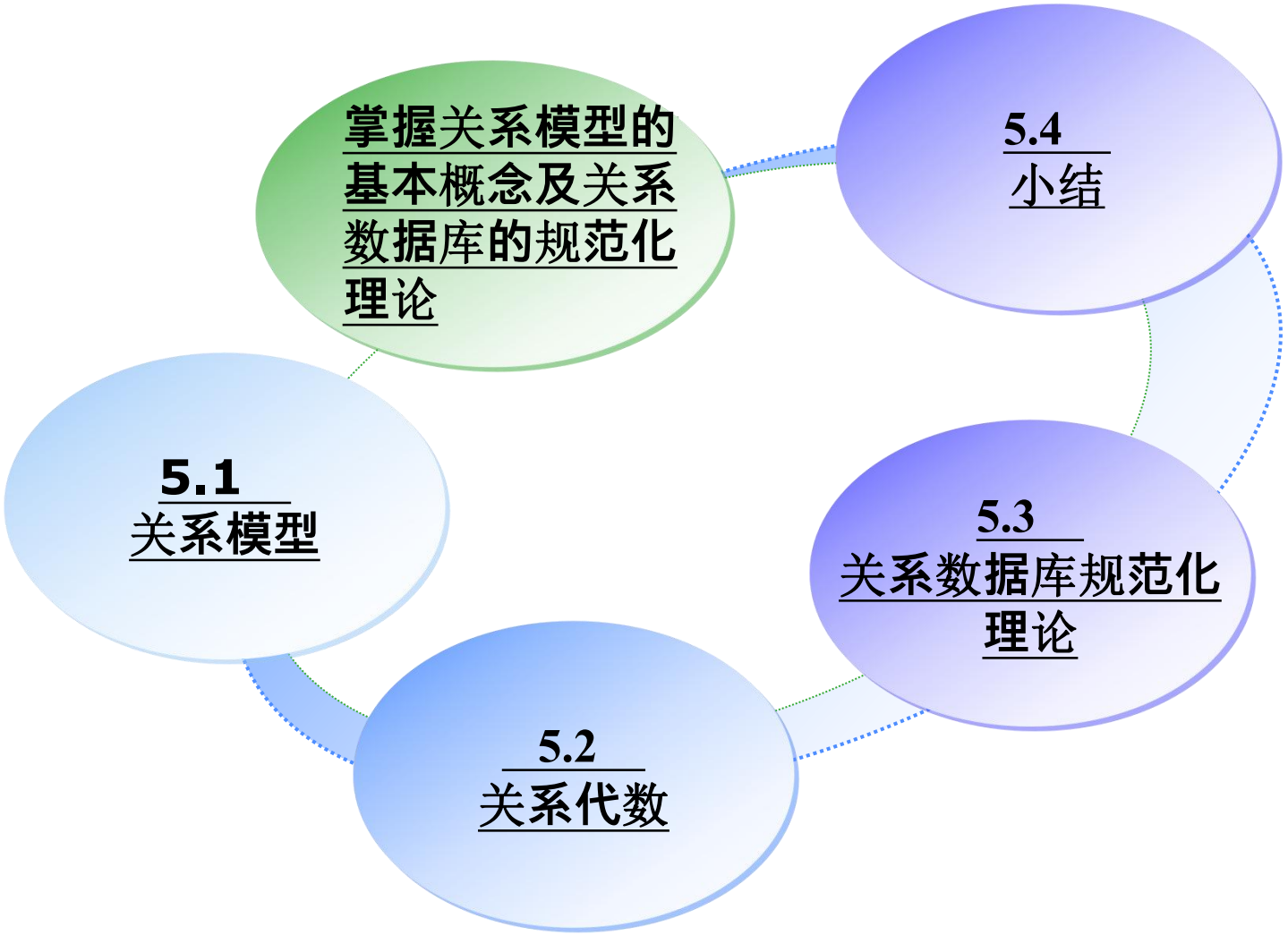
数据库系统

第五章 关系数据库基本理论

北京工业大学耿丹学院

计算机科学与技术专业

本章概述


掌握关系模型的基本概念及关系数据库的规范化理论
5.4
小结
5.1
关系模型
5.3
关系数据库规范化理论
5.2
关系代数

5.1 关系模型

- 关系数据结构
- 关系数据操作
- 关系的完整性约束

5.1.1 关系数据结构

- 关系模型建立在集合代数的基础上
- 关系数据结构的基本概念
  - 关系
  - 关系模式
  - 关系数据库

一、关系

1. 域（Domain）
2. 笛卡尔积（Cartesian Product）
3. 关系（Relation）

5.1.1 关系数据结构

一、关系

域（Domain）

- 一组值的集合，这组值具有相同数据类型。

例如：整数集合，实数集合，字符串集合，{男, 女} 等都是域。

- 基数：域中元素的个数称为域的基数。

D_1={教授, 副教授, 讲师, 助教}，表示职称的集合；
其中D_1的基数是4

5.1.1 关系数据结构

一、关系

笛卡尔积（Cartesian Product）

1） 笛卡尔积的定义

给定一组域D_1，D_2，…，D_n，这些域中可以有相同的。D_1，D_2，…，D_n的笛卡尔积为：

$$D_1 \times D_2 \times \ldots \times D_n = \{ (d_1, d_2, \ldots, d_n) \mid d_i \in D_i, \ i=1, 2, \ldots, n \}$$

- 所有域的所有取值的一个组合
- 不能重复

5.1.1 关系数据结构

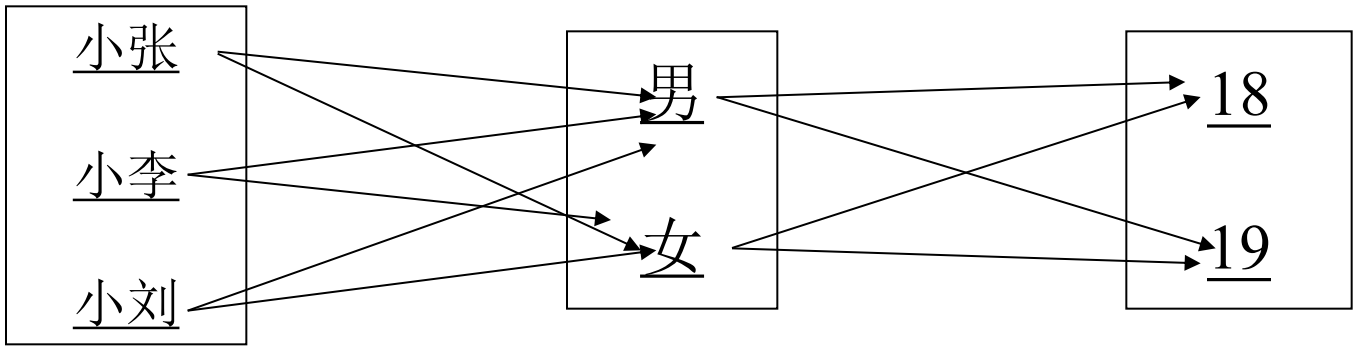
例：

D_1= 姓名 = { 小张，小李，小刘 }

D_2= 性别 = {男，女}

D_3= 年龄 = {18，19}

求D_1，D_2，D_3的笛卡尔积？



$D1 \times D2 \times D3$ =

{(小张，男，18)，(小张，男，19)，(小张，女，18)，(小张，女，19)，
(小李，男，18)，(小李，男，19)，(小李，女，18)，(小李，女，19)，
(小刘，男，18)，(小刘，男，19)，(小刘，女，18)，(小刘，女，19)}

笛卡尔积（Cartesian Product）

2) 元组（Tuple）

笛卡尔积中每一个元素（d_1，d_2，…，d_n）叫作一个n元组（n-tuple）或简称元组。

3) 分量（Component）

笛卡尔积元素（d_1，d_2，…，d_n）中的每一个值d_i叫作一个分量。

4) 基数（Cardinal number）

若D_i（i=1，2，…，n）为有限集，其基数为m_i（i=1，2，…，n），则$D_1 \times D_2 \times \ldots \times D_n$的基数M为：

$$M = \prod_{i=1}^{n} m_i$$

笛卡尔积（Cartesian Product）

5)笛卡尔积的表示方法

笛卡尔积可表示为一个二维表。表中的每行对应一个元组，表中的每列对应一个域。

在上述例中，12个元组可列成一张二维表

表1：

姓名	性别	年龄
小张	男	18
小张	男	19
小张	女	18
小张	女	19
小李	男	18
小李	男	19
小李	女	18
小李	女	19
小刘	男	18
小刘	男	19
小刘	女	18
小刘	女	19

5.1.1 关系数据结构

关系（Relation）

1) 关系的定义

笛卡尔积$D_1 \times D_2 \times \ldots \times D_n$的子集叫作在域$D_1$，$D_2$，…，$D_n$上的关系。可表示为:

$R（D_1，D_2，\ldots，D_n）$

***R*：关系名**

***n*：关系的目或度（Degree）**

- 关系是笛卡尔积的有限子集。
- 关系是一个二维表。

5.1.1 关系数据结构

在表1的笛卡尔积中取出有意义的元组来构造一个学生关系：

学生(姓名，性别，年龄)

- 假设小张和小李是男生且均为18岁，小刘是女生，19岁。

表2：学生关系

姓名	性别	年龄
小张	男	18
小李	男	18
小刘	女	19

表1： D_1，D_2，D_3的笛卡尔积

姓名	性别	年龄
小张	男	18
小张	男	19
小张	女	18
小张	女	19
小李	男	18
小李	男	19
小李	女	18
小李	女	19
小刘	男	18
小刘	男	19
小刘	女	18
小刘	女	19

5.1.1 关系数据结构

关系（Relation）

2) 元组

- 关系中的每一行称作一个元组
- 组成元组的元素为分量。
- 如表2中有三个元组

表2：学生关系

姓名	性别	年龄
小张	男	18
小李	男	18
小刘	女	19

3) 属性

- 关系中的每一列称为一个属性。
- 如表2中有三个属性，分别为：姓名、性别和年龄。
- 关系中的属性名具有标识列的作用，则同一关系中的属性名（列名）不能相同。

5.1.1 关系数据结构

关系（Relation）

4) 码

- 码：在关系中唯一标识元组的最小的属性集称为该关系的码或关键字。

- 候选码：一个关系中可能有若干个码，它们称为该关系的候选码或候选关键字。

  - ✓ 任何候选码中的属性为主属性；
  - ✓ 不包含在候选码中的属性为非主属性

- 主码：当一个关系有多个候选码时，应选定其中的一个候选码为主码。一般主码也简称码。

学生关系

学号	姓名	性别	年龄
05801	小张	男	18
05802	小李	男	18
05803	小刘	女	19

学生选课关系

学号	课程名	课时
05801	C	48
05801	数据库	56
05802	C	48

5.1.1 关系数据结构

- 外码：如果关系A中的某属性集是关系B的码，但不是A的码，则称该属性集为A的外码或外关键字。

A：学生登记表

学号	姓名	学院编号
05801	小张	01
05802	小李	02
05803	小王	02

B: 学院登记表

学院编号	学院名称
01	信息
02	经管
03	自动化

关系A*称为*参照关系　　关系B*称为*被参照关系或目标关系。

说明：

- 关系*A*和*B*不一定是不同的关系。
- 目标关系*B*的主码和参照关系A的外码必须定义在同一个（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，往往取相同的名字，以便于识别。

5.1.1 关系数据结构

- 全码：

✓ 若关系中的候选码只包含一个属性，则称它为单属性码。

✓ 若候选码是由多个属性构成，则称它为多属性码。

✓ 若关系中只有一个候选码，且这个候选码中包括全部属性，则该候选码称为全码。

学生选课关系

学号	课程名
05801	C
05801	数据库
05802	C

5.1.1 关系数据结构

6) 数据库中基本关系的性质

① 同一属性的数据具有同质性

✓ 每一列中的分量是同一类型的数据，来自同一个域。

✓ 例如：学生选课表的结构中：选课（学号，课号，成绩），而成绩的属性值不能有百分制、5分制、或“优”“良”等多种取值法。

② 同一关系的属性名具有不能重复性

✓ 同一关系中不同属性的数据可出自同一个域，但是不同的属性要给予不同的属性名。

✓ 如：要设计一个能存储两科成绩的学生成绩表，其表结构不能写为：学生成绩（学号，成绩，成绩）

可以设计为：学生成绩（学号，成绩1，成绩2）

③ 关系中的列位置具有顺序无关性

✓ 关系中列的次序可以任意交换、重新组织，属性顺序不影响使用。

5.1.1 关系数据结构

④ 关系中的元组具有无冗余性

✓ 关系中的任意两个元组不能完全相同。

⑤ 关系中的元组位置具有顺序无关性

✓ 关系元组的顺序可以任意交换。

⑥ 关系中每一个分量都必须是不可分的数据项

✓ 关系规范条件中最基本的一条就是关系的每一个分量必须是不可分的数据项，即分量是原子量。

姓名	所在系	成绩	
		C成绩	数据结构成绩
刘克	计算机	89	90
李明	经管	86	88

姓名	所在系	C成绩	数据结构成绩
刘克	计算机	89	90
李明	经管	86	88

二、关系模式

关系模式的定义

- 关系的描述称为关系模式。关系模式可以形式化地表示为：

$$R（U，D，\mathrm{dom}，F）$$

其中：

R：关系名；

U：组成该关系的属性名集合；

D：属性集*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

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。

二、关系模式

关系模式的定义

- 关系模式通常可以简记为

$R(U)$ 或 $R(A_1, A_2, \ldots An)$

其中：R 关系名

$A_1, A_2, \ldots, A_n$ 属性名

U 是属性的集合

$U=\{A_1, A_2, \ldots, A_n\}$

二、关系模式

关系模式与关系

- 关系模式是对关系的描述，是关系的型，即框架或结构。是静态的，稳定的。
- 关系是按关系模式组织的表格。是关系模式在某一时刻的状态或内容。是动态的。

关系模式和关系往往统称为关系，通过上下文加以区别

三、关系数据库

关系数据库的定义

在一个给定的应用领域中，所有实体及实体之间联系所形成的关系的集合构成一个关系数据库。

关系数据库的型和值

关系数据库的型称为关系数据库模式，是对关系数据库的描述。

关系数据库的值是这些关系模式在某一时刻对应的关系的集合，也就是所说的关系数据库的数据。

5.1.2 关系操作

一、基本的关系操作

- 查询

  选择、投影、连接、除、并、交、差

- 更新

  插入、删除、修改

- 查询的表达能力是其中最主要的部分

关系操作的特点：集合操作方式

即操作的对象和结果都集合。

5.1.2 关系操作

二、关系数据语言的分类

- 关系代数语言
  - 对关系的运算是用代数方式来表达查询要求
- 关系演算语言：用谓词演算（逻辑方式）来表达查询要求的查询语言

- 上述两种的特点：
  - 抽象
  - 常用评估实际系统查询语言能力的标准或理论基础

- 具有关系代数和关系演算双重特点的语言
  - 典型代表：**SQL**（结构化查询语言）

5.1.3 关系的完整性

一、关系的三类完整性约束

关系模型的完整性规则是对关系的某种约束条件。

关系模型中三类完整性约束：

实体完整性

参照完整性

用户定义的完整性

实体完整性和参照完整性是关系模型必须满足的完整性约束条件，被称作是关系的两个不变性，应该由关系系统自动支持。

用户定义的完整性是应用领域需要遵循的约束条件。

5.1.3 关系的完整性

二、 实体完整性

1、实体完整性规则（Entity Integrity）：

关系数据库中所有的表都必须有主码，要求主码不能为空，且不能取相同的值。

例： 学生(学号，姓名，性别，年龄)

学号属性为主码，则其不能取空值

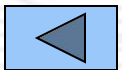
2、说明：

例如：

实体完整性规则规定基本关系的主码不能取空值。

例：选修（学号，课程号，成绩）

“学号、课程号”为主码，则两个属性都不能取空值。



5.1.3 关系的完整性

三、 参照完整性

1. 关系间的引用

学生

学号	姓名	性别	专业号	年龄
801	张三	女	01	19
802	李四	男	01	20
803	王五	男	01	20
804	赵六	女	02	20
805	钱七	男	02	19

在关系模型中实体及实体间的联系都是
存在着关系与关系间的引用。

例1 学生实体、专业实体以及专业与学生间的一对多联系。

学生（学号，姓名，性别，专业号，年龄）

专业（专业号，专业名）

专业

专业号	专业名
01	信息
02	数学
03	计算机

学生关系中每个元组的“专业号”属性只取下面两类值：

（1）空值，表示尚未给该学生分配专业

（2）非空值，这时该值必须是专业关系中某个元组的“专业号”值，表示该学生不可能分配到一个不存在的专业中。

5.1.3 关系的完整性

例2 学生、课程、学生与课程之间的多对 多

学生（学号，姓名，性别，专业号，年龄）

课程（课程号，课程名，学分）

选修（学号，课程号，成绩）

选修（学号，课程号，成绩）

“学号”和“课程号”是选修关系中的主属性按照实体完整性和参照完整性规则，它们只能取相应被参照关系中已经存在的主码值。

学生

学号	姓名	性别	专业号	年龄
801	张三	女	01	19
802	李四	男	01	20
803	王五	男	01	20
804	赵六	女	02	20
805	钱七	男	02	19

课程

课程号	课程名	学分
01	数据库	4
02	数据结构	4
03	编译	4
04	PASCAL	2

学生选课

学号	课程号	成绩
801	04	92
801	03	78
801	02	85
802	03	82
802	04	90
803	04	88

5.1.3 关系的完整性

例3 学生实体及其内部的领导联系(一对多)

学生（学号，姓名，性别，专业号，年龄，班长）

学生

学号	姓名	性别	专业号	年龄	班长
801	张三	女	01	19	802
802	李四	男	01	20	802
803	王五	男	01	20	802
804	赵六	女	02	20	805
805	钱七	男	02	19	805

学生（学号，姓名，性别，专业号，年龄，班长）

“班长”属性值可以取两类值：

（1）空值，表示该学生所在班级尚未选出班长，或该学生本人即是班长；

（2）非空值，这时该值必须是本关系中某个元组的学号值。

5.1.3 关系的完整性

2. 参照完整性规则

若属性（或属性组）***F***是关系***R***的外码，它与关系***S***的主码相对应（关系***R***和***S***不一定是不同的关系），则对于***R***中每个元组在***F***上的值必须为：

- 或者取空值（***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。

5.1.3 关系的完整性

四、 用户定义的完整性

用户定义的完整性是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，以便用统一的系统的方法处理它们，而不要由应用程序承担这一功能。

例：课程(课程号，课程名，学分)

- “课程号”属性必须取唯一值
- 非主属性“课程名”也不能取空值
- “学分”属性只能取值{1，2，3，4，5}

5.2 关系代数

一、概述

1. 关系代数

一种抽象的查询语言对关系的运算来表达查询

2. 运算的三要素：　运算对象、运算结果、运算符

3. 关系代数运算的三个要素

运算对象：关系
运算结果：关系
运算符：四类

4. 关系代数运算的分类

- 传统的集合运算：并、差、交、广义笛卡尔积
- 专门的关系运算：选择、投影、连接、除

关系代数运算符

1. **集合运算符**
   - 将关系看成元组的集合
   - 运算是从关系的“水平”方向即行的角度来进行
2. **专门的关系运算符**

   不仅涉及行而且涉及列
3. **算术比较符**

   辅助专门的关系运算符进行操作
4. **逻辑运算符**

   辅助专门的关系运算符进行操作

5.2 关系代数

表1 关系代数运算符

运算符		含义	运算符		含义
集合运算符	∪	并	比较运算符	>	大于
	-	差		≥	大于等于
	∩	交		<	小于
	×	广义笛卡尔积		≤	小于等于
				= ≠	等于 不等于
专门的关系运算符	σ	选择	逻辑运算符	¬	非
	π	投影		∧	与
	⋈	连接		∨	或
	÷	除			

二、传统的集合运算

- 并　$R \cup S = \{ t \mid t \in R \vee t \in S \}$
- 差　$R - S = \{ t \mid t \in R \wedge t \notin S \}$
- 交　$R \cap S = \{ t \mid t \in R \wedge t \in S \}$　或　$R \cap S = R - (R - S)$

要求R和S

- 具有相同的目n（即两个关系都有n个属性）
- 相应的属性取自同一个域

- 广义笛卡尔积

$$R \times S = \{ \widehat{t_r\ t_s} \mid t_r \in R \wedge t_s \in S \}$$

R (n目关系，k_1个元组)　　S (*m*目关系，k_2个元组)

R×*S*

列：(n+m)列的元组的集合；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：$k_1 \times k_2$个元组

5.2 关系代数

R

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

S

A	B	C
a1	b2	c2
a1	b3	c2
a2	b2	c1

$R \cup S$

A	B	C
a1	b1	c1
a1	b2	c2
a1	b3	c2
a2	b2	c1

R-S

A	B	C
a1	b1	c1

$R \cap S$

A	B	C
a1	b2	c2
a2	b2	c1

$R \times S$

R.A	R.B	R.C	S.A	S.B	S.C
a1	b1	c1	a1	b2	c2
a1	b1	c1	a1	b3	c2
a1	b1	c1	a2	b2	c1
a1	b2	c2	a1	b2	c2
a1	b2	c2	a1	b3	c2
a1	b2	c2	a2	b2	c1
a2	b2	c1	a1	b2	c2
a2	b2	c1	a1	b3	c2
a2	b2	c1	a2	b2	c1

三、 专门的关系运算

- 选择
- 投影
- 连接
- 除

1. 选择（Selection）

1) 选择又称为限制（Restriction）

2) 选择运算符的含义

在关系***R***中选择满足给定条件的诸元组。

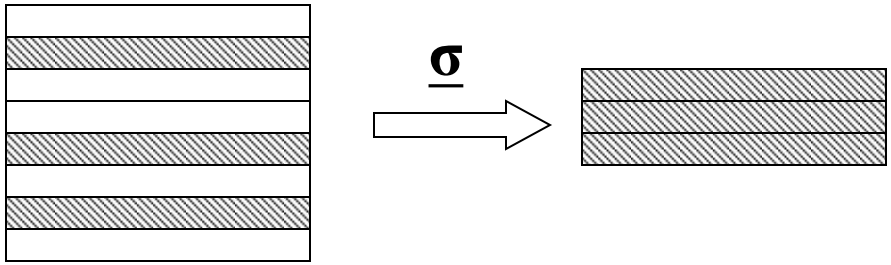
$$\sigma_F(R) = \{t \mid t \in R \wedge F(t) = \text{'真'}\}$$

σ为选择运算符

F：选择条件，是一个逻辑表达式，由运算对象（属性名、常数、简单函数）、算术运算符和逻辑运算符连接。

1. 选择（Selection）

3) 选择运算是从行的角度进行的运算



4) 举例

5.2 关系代数

4)举例：设有一个学生-课程数据库，包括学生关系Student、课程关系Course和选修关系SC。

Student

学 号 Sno	姓 名 Sname	性 别 Ssex	年 龄 Sage	所在系 Sdept
01	李勇	男	20	CS
02	刘晨	女	19	IS
03	王敏	女	18	MA
04	张立	男	19	IS

Course

课程号 Cno	课程名 Cname	先行课 Cpno	学分 Ccredit
1	数据库	5	4
2	数学		2
3	信息系统	1	4
4	操作系统	6	3
5	数据结构	7	4
6	数据处理		2
7	C语言	6	4

SC

学 号 Sno	课 程 号 Cno	成 绩 Grade
01	1	92
01	2	85
01	3	88
02	2	90
02	3	80

[例1] 查询信息系（IS系）全体学生

[例2] 查询年龄小于20岁的学生

5.2 关系代数

[例1] 查询信息系（IS系）全体学生

$\sigma_{Sdept = 'IS'}(Student)$

或 $\sigma_{5 ='IS'}(Student)$

Sno	Sname	Ssex	Sage	Sdept
02	刘晨	女	19	IS
04	张立	男	19	IS

[例2] 查询年龄小于20岁的学生

$\sigma_{Sage < 20}(Student)$

或 $\sigma_{4 < 20}(Student)$

Sno	Sname	Ssex	Sage	Sdept
02	刘晨	女	19	IS
03	王敏	女	18	MA
04	张立	男	19	IS

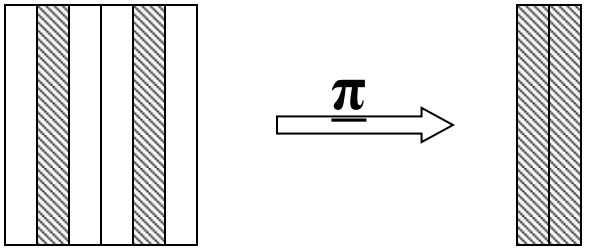
2. 投影（Projection）

1）投影运算符的含义

从*R*中选择出若干属性列组成新的关系

$$\pi_A(R) = \{ t[A] \mid t \in R \}$$

A：*R*中的属性列



2）投影操作主要是从列的角度进行运算

注：但投影之后不仅取消了原关系中的某些列，而且还可能取消某些元组（避免重复行）

3）举例

[例3] 查询学生的姓名和所在系

即求**Student**关系上学生姓名和所在系两个属性上的投影

$\Pi_{Sname,\ Sdept}(Student)$

或 $\Pi_{2,\ 5}(Student)$

结果：

Sname	Sdept
李勇	CS
刘晨	IS
王敏	MA
张立	IS

[例4] 查询学生关系Student中都有哪些系

$\pi_{Sdept}(Student)$

结果：

Sdept
CS
IS
MA

3. 连接（Join）

1）连接也称为θ连接

2）连接运算的含义

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

$$R \underset{A\theta B}{\bowtie} S = \{ \widehat{t_r t_s} \mid t_r \in R \wedge t_s \in S \wedge t_r[A]\theta t_s[B] \}$$

A和B：分别为R和S上度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$R \times S$中选取（R关系）在A属性组上的值与（S关系）在B属性组上值满足比较关系的元组。

5.2 关系代数

[例5]

R

A	B	C
a_1	b_1	5
a_1	b_2	6
a_2	b_3	8
a_2	b_4	12

S

B	E
b_1	3
b_2	7
b_3	10
b_3	2
b_5	2

$R \times S$

A	R.B	C	S.B	E
a_1	b_1	5	b_1	3
a_1	b_1	5	b_2	7
a_1	b_1	5	b_3	10
a_1	b_1	5	b_3	2
a_1	b_1	5	b_5	2
a_1	b_2	6	b_1	3
a_1	b_2	6	b_2	7
a_1	b_2	6	b_3	10
a_1	b_2	6	b_3	2
a_1	b_2	6	b_5	2
a_2	b_3	8	b_1	3
…	…	…	…	…

$R \underset{C<E}{\bowtie} S$

A	R.B	C	S.B	E
a_1	b_1	5	b_2	7
a_1	b_1	5	b_3	10
a_1	b_2	6	b_2	7
a_1	b_2	6	b_3	10
a_2	b_3	8	b_3	10

3. 连接（Join）

3）两类常用连接运算

等值连接（equijoin）

θ为“＝”的连接运算称为等值连接

等值连接的含义：

从关系R与S的广义笛卡尔积中选取A、B属性值相等的那些元组，即等值连接为：

$$R \underset{A=B}{\bowtie} S = \{ t_r \widehat{\ } t_s \mid t_r \in R \wedge t_s \in S \wedge t_r[A] = t_s[B] \}$$

自然连接（Natural join）

自然连接是一种特殊的等值连接

两个关系中进行比较的分量必须是相同的属性组,在结果中把重复的属性列去掉。

自然连接的含义：（R和S具有相同的属性组B）

$$R \bowtie S = \{ t_r \widehat{\ } t_s \mid t_r \in R \wedge t_s \in S \wedge t_r[B] = t_s[B] \}$$

5.2 关系代数

[例6]

R

A	B	C
a_1	b_1	5
a_1	b_2	6
a_2	b_3	8
a_2	b_4	12

S

B	E
b_1	3
b_2	7
b_3	10
b_3	2
b_5	2

$R \times S$

A	R.B	C	S.B	E
a_1	b_1	5	b_1	3
a_1	b_1	5	b_2	7
a_1	b_1	5	b_3	10
a_1	b_1	5	b_3	2
a_1	b_1	5	b_5	2
a_1	b_2	6	b_1	3
a_1	b_2	6	b_2	7
a_1	b_2	6	b_3	10
a_1	b_2	6	b_3	2
a_1	b_2	6	b_5	2
a_2	b_3	8	b_1	3
…	…	…	…	…

等值连接 $R \bowtie S$ $R.B=S.B$

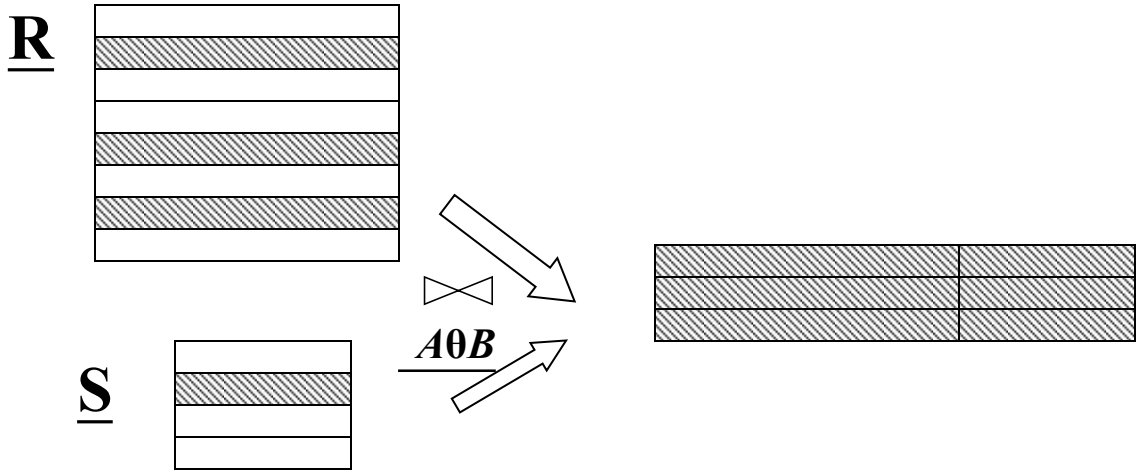
A	R.B	C	S.B	E
a_1	b_1	5	b_1	3
a_1	b_2	6	b_2	7
a_2	b_3	8	b_3	10
a_2	b_3	8	b_3	2

自然连接 $R \bowtie S$

A	B	C	E
a_1	b_1	5	3
a_1	b_2	6	7
a_2	b_3	8	10
a_2	b_3	8	2

3. 连接（Join）

4）一般的连接操作是从行的角度进行运算。



自然连接还需要取消重复列，所以是同时从行和列的角度进行运算。

外连接

- 外连接(Outer join): 在自然连接中把舍弃的元组也保存在结果关系中,其它属性上填空值(Null)
- 左连接(Left outer join): 只保留左边关系要舍去的元组
- 右连接(Right outer join): 只保留右边关系要舍去的元组

A	B	C	E
a_1	b_1	5	3
a_1	b_2	6	7
a_2	b_3	8	10
a_2	b_3	8	2
a_2	b_4	12	Null
Null	b_5	Null	2

(a) R、S外连接

A	B	C	E
a_1	b_1	5	3
a_1	b_2	6	7
a_2	b_3	8	10
a_2	b_3	8	2
a_2	b_4	12	Null

(b) R、S左连接

A	B	C	E
a_1	b_1	5	3
a_1	b_2	6	7
a_2	b_3	8	10
a_2	b_3	8	2
Null	b_5	Null	2

(c) R、S右连接

4. 综合举例

以学生-课程数据库为例,数据库中包含3个关系:

S(Sno, Sname, Sage,Ssex,Sdept)

SC(Sno,Cno,Grade);

C(Cno,Cname,Cpno,Ccredit)

[例1] 检索学习课程号为2的学生学号和成绩

$$\pi_{Sno，Grade}(\sigma_{Cno='2'}(SC))$$

或$\pi_{1，3}(\sigma_{2='2'}(SC))$

[例2] 检索学习课程号为2的学生学号和姓名

$$\pi_{Sno，Sname}(\sigma_{Cno='2'}(S \bowtie SC))$$

[例 3] 检索课程名为数据结构的学生学号与姓名

$$\pi_{Sno,Sname}(\sigma_{Cname='数据结构'}(C \bowtie SC \bowtie S))$$

[例4] 检索选修课程号为2或4的学生学号

$$\Pi_{Sno}(\sigma_{Cno='2' \vee Cno='4'}(SC))$$

[例5] 检索不选修课程号为2的学生姓名与年龄

$$\pi_{Sname,\ Sage}(S) - \pi_{Sname,\ Sage}(\sigma_{Cno='2'}(S \bowtie SC))$$

关系模型和关系代数小结

- 理解关系、关系模式、关系数据库的形式化定义
- 掌握码的相关概念：码，候选码、主码、外码、全码
- 掌握关系的三类完整性约束
- 掌握关系代数的常见运算，能熟练使用关系代数表达式来表达查询。

5.3.1 问题的提出

一、概念回顾
二、关系模式的形式化定义
三、什么是数据依赖
四、关系模式的简化定义
五、数据依赖对关系模式影响

一、概念回顾

- 关系：

  描述实体、属性、实体间的联系。

  - 从形式上看，它是一张二维表，是所涉及属性的笛卡尔积的一个子集。

- 关系模式：
- 用来定义关系。
- 关系数据库：
- 基于关系模型的数据库，利用关系来描述现实世界。
  - 从形式上看，它由一组关系组成。
- 关系数据库的模式：
- 定义这组关系的关系模式的全体。

二、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

关系模式的简化表示

关系模式R（U, D, DOM, F）

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

三、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

- 是通过一个关系中属性间值的相等与否体现出来的数据间的相互关系；
- 是现实世界属性间相互联系的抽象，是数据内在的性质，是语义的体现。

3. 数据依赖的主要类型

- 函数依赖（**Functional Dependency**，简记为**FD**）
- 多值依赖（**Multivalued Dependency**，简记为**MVD**）

四、数据依赖对关系模式的影响

例：描述学校数据库：

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、F>**

U ＝｛ Sno, Sdept, Mname, Cname, Grade ｝

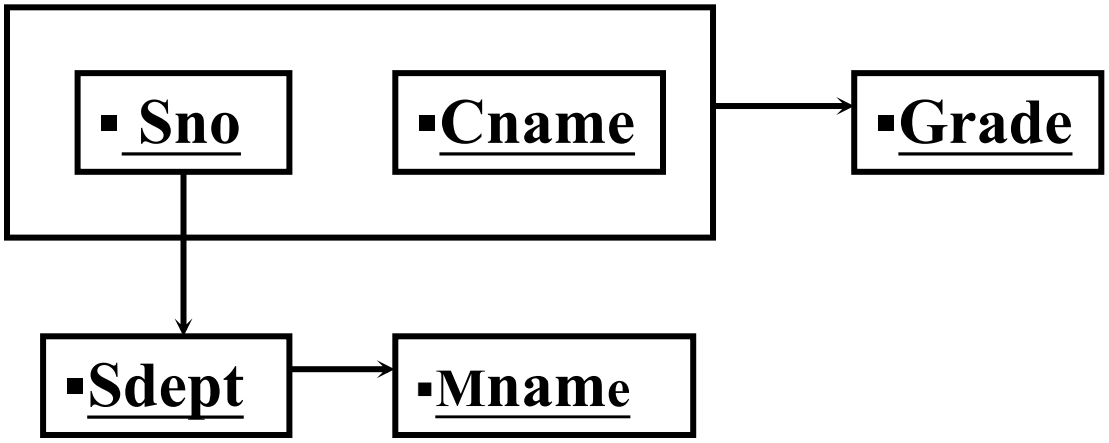
学校数据库的语义：

1. 一个系有若干学生， 一个学生只属于一个系；
2. 一个系只有一名主任；
3. 一个学生可以选修多门课程， 每门课程有若干学生选修；
4. 每个学生所学的每门课程都有一个成绩。

五、数据依赖对关系模式的影响（续）

属性组U上的一组函数依赖F：

$$F = \{ Sno \rightarrow Sdept,\ Sdept \rightarrow Mname,\ (Sno, Cname) \rightarrow Grade \}$$



▪某一时刻关系模式Student<U,F>的一个实例：

Sno	Sdept	Mname	Cname	Grade
S1	信息系	王主任	数据库	90
S2	信息系	王主任	数据库	87
S3	信息系	王主任	数据库	99
S4	信息系	王主任	数据库	80
S5	信息系	王主任	数据库	88
…	…	…	…	…

▪存在的问题：

- 1. 数据冗余太大——浪费大量的存储空间

2. 更新异常（Update Anomalies）

数据冗余，更新数据时，维护数据完整性代价大。

▪例：每一个系主任的姓名重复出现

例：某系更换系主任后，系统必须修改与该系学生有关的每一个元组。

- 某一时刻关系模式Student<U,F>的一个实例：

Sno	Sdept	Mname	Cname	Grade
S1	信息系	王主任	数据库	90
S2	信息系	王主任	数据库	87
S3	信息系	王主任	数据库	99
S4	信息系	王主任	数据库	80
S5	信息系	王主任	数据库	88
…	…	…	…	…

- 存在的问题：

3. 插入异常（Insertion Anomalies）

该插的数据插不进去

例，如果一个系刚成立，尚无学生，我们就无法把这个系及其系主任的信息存入数据库。

4. 删除异常（Deletion Anomalies）

不该删除的数据不得不删

例，如果某个系的学生全部毕业了，我们在删除该系学生信息的同时，把这个系及其系主任的信息也丢掉了。

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。

- 分解成三个关系模式：
- S (Sno, Sdept, Sno → Sdept);
- SC (Sno, Cname, Grade, (Sno, Cname) → Grade);
- DEPT (Sdept, Mname, Sdept→Mname);

5.3.2 规范化

规范化理论正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、删除异常、更新异常和数据冗余问题。

5.3.2 规范化

- 函数依赖
- 范式
- 规范化总结
- 例子

一、函数依赖

定义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。

X称为这个函数依赖的决定属性集(决定因素)。　　Y=f(x)

说明：1. 函数依赖指R的所有关系实例均要满足的约束条件。

2. 函数依赖是语义范畴的概念。只能根据数据的语义来确定函数依赖。例如“姓名→年龄”函数依赖只在不允许有同名的条件下成立

3. 数据库设计者可以对现实世界作强制的规定。例如规定不允许同名人出现，函数依赖“姓名→年龄”成立。所插入的元组必须满足规定的函数依赖，若有同名，则拒绝插入该元组。

例: Student(Sno, Sname, Ssex, Sage, Sdept)

假设不允许重名，则有:

Sno → Ssex，　Sno → Sage , Sno → Sdept，

Sno ←→ Sname, Sname → Ssex，　Sname → Sage

Sname → Sdept

但Ssex $\not\rightarrow$ Sage

若X→Y，并且Y→X，则记为X←→Y。

若Y不函数依赖于X，则记为X$\not\rightarrow$Y。

二、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 ⊆ X，则称X→Y是平凡的函数依赖

若X→Y，但Y ⊈ X，则称X→Y是非平凡的函数依赖

对于任一关系模式，平凡函数依赖都是必然成立的，它不反映新的语义，若不特别声明，总是讨论非平凡函数依赖。

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 (Sno, Cno) → Cno

三、完全函数依赖与部分函数依赖

定义2 在关系模式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完全函数依赖于X，记作 $X \xrightarrow{F} Y$。

若X→Y，但Y不完全函数依赖于X，则称Y部分函数依赖于X，记作 $X \xrightarrow{P} Y$。

例：在关系SC(Sno, Cno, Grade)中，

由于：Sno →Grade，Cno → Grade，

因此：(Sno, Cno) $\xrightarrow{F}$ Grade

- 而在Student(Sno,Sdept,Cno,Mname,Grade)中
- (Sno,Cno)→Sdept
- 是部分函数依赖
- 因为Sno →Sdept成立，且Sno是（Sno，Cno）的真子集

四、传递函数依赖

定义3　在关系模式R(U)中，如果X→Y，Y→Z，且 $Y \subseteq X$，$Y \not\to X$，则称Z传递函数依赖于X。

记为：$X \xrightarrow{传递} Z$

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

五、码

定义 4 设K为关系模式R<U,F>中的属性或属性组合。若 $K \xrightarrow{F} U$，则K称为R的一个侯选码（Candidate Key）。若关系模式R有多个候选码，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；
  - 不包含在任何码中的属性称为非主属性
- 全码 (ALL KEY)：整个属性组是码

[例]

关系模式S(Sno,Sdept,Sage)，单个属性Sno是码，

SC（Sno，Cno，Grade）中，（Sno，Cno）是码

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：

$$1NF \supset 2NF \supset 3NF \supset BCNF \supset 4NF \supset 5NF$$

- 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

一、1NF

- 定义

  如果一个关系模式R的所有属性都是不可分的基本数据项，则R∈1NF。

- 第一范式是对关系模式的最起码的要求。不满足第一范式的数据库模式不能称为关系数据库。
- 但是满足第一范式的关系模式并不一定是一个好的关系模式。

例: 关系模式 SLC(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。SLC的码为 (Sno, Cno)

函数依赖包括：

(Sno, Cno) $\xrightarrow{F}$ Grade

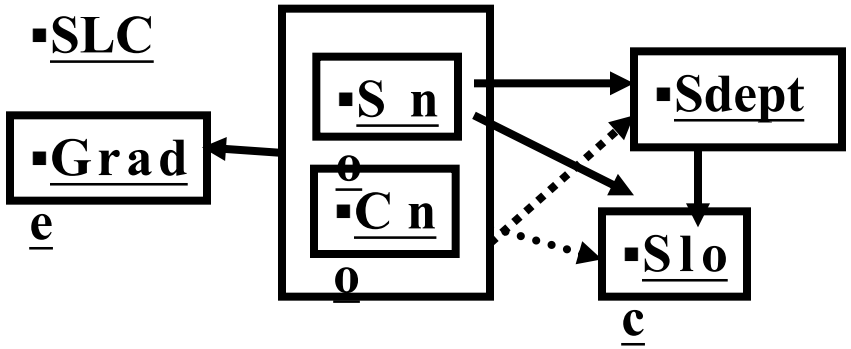
Sno → Sdept

(Sno, Cno) $\xrightarrow{P}$ Sdept

Sno → Sloc

(Sno, Cno) $\xrightarrow{P}$ Sloc

Sdept → Sloc



SLC满足第一范式。

非主属性Sdept和Sloc部分函数依赖于码(Sno, Cno)

SLC (Sno, Sdept, Sloc, Cno, Grade)不是一好的关系模式

(1) 插入异常

假设**Sno＝02，Sdept＝IS，Sloc＝N**的学生还未选课，因课程号是主属性，因此该学生的信息无法插入**SLC**。

(2) 删除异常

假定某个学生本来只选修了**3**号课程这一门课。现在因身体不适，他连**3**号课程也不选修了。因课程号是主属性，此操作将导致该学生信息的整个元组都要删除。

(3) 数据冗余度大

如果一个学生选修了**10**门课程，那么他的**Sdept**和**Sloc**值就要重复存储了**10**次。

(4) 修改复杂

例如学生转系，在修改此学生元组的**Sdept**值的同时，还可能需要修改住处（**Sloc**）。如果这个学生选修了**n**门课，则必须无遗漏地修改**n**个元组中全部**Sdept**、**Sloc**信息。

❖ 原因

Sdept、 Sloc部分函数依赖于码。

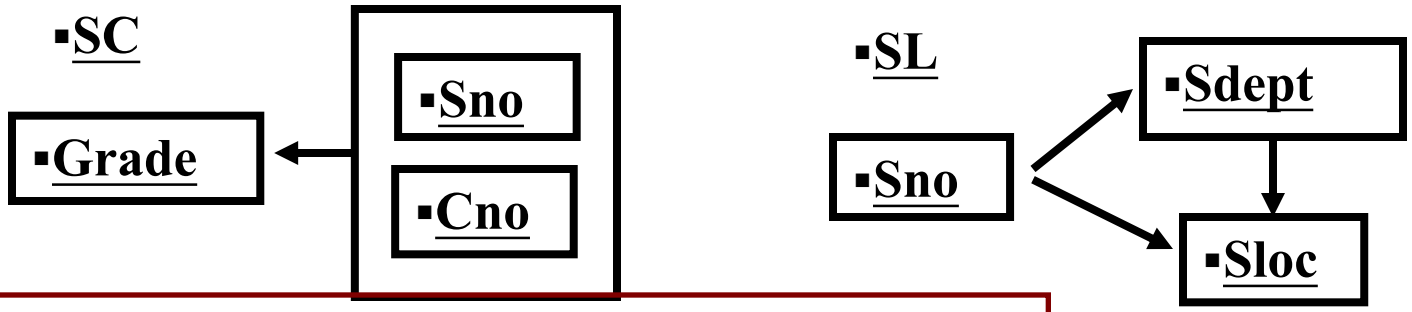
❖ 解决方法

SLC分解为两个关系模式，以消除这些部分函数依赖

SC（Sno， Cno， Grade）

SL（Sno， Sdept， Sloc）

❖ 函数依赖图：



❖关系模式SC的码为（Sno，Cno）

❖ 关系模式SL的码为Sno

❖这样非主属性对码都是完全函数依赖

二、2NF

定义 若关系模式R∈1NF，并且每一个非主属性都完全函数依赖于R的码，则R∈2NF。

例：SLC(Sno, Sdept, Sloc, Cno, Grade) ∈1NF

SLC(Sno, Sdept, Sloc, Cno, Grade) ∉ 2NF

SC（Sno， Cno， Grade） ∈ 2NF

SL（Sno， Sdept， Sloc） ∈ 2NF

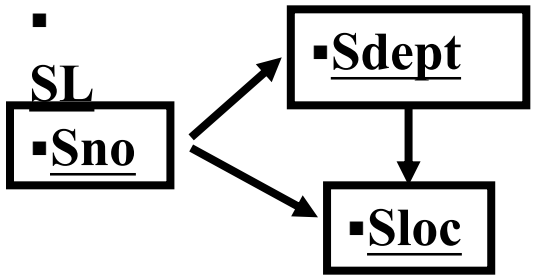
- 采用模式分解法将一个1NF的关系分解为多个2NF的关系，可以在一定程度上减轻原1NF关系中存在的插入异常、删除异常、数据冗余度大、修改复杂等问题。
- 将一个1NF关系分解为多个2NF的关系，并不能完全消除关系模式中的各种异常情况和数据冗余。

- 模式分解：就是将一个关系模式分解为若干个模式，分解后的模式具有下面的三个特征：
  - 分解后的模式均为高一级的模式
  - 分解后关系中的数据不会丢失，即分解后的关系再经连接后能恢复到原来的关系，称为无损连接
  - 分解后关系中的函数依赖不会丢失，这叫做依赖保持。

例：2NF关系模式SL(Sno, Sdept, Sloc)中

函数依赖： Sno→Sdept Sno→Sloc，
Sdept→Sloc

Sloc传递函数依赖于Sno，即SL中存在非主属性对码的传递函数依赖。



解决方法：

采用模式分解法，把SL分解为两个关系模式，以消除传递函数依赖：

- SD（Sno， Sdept）
- DL（Sdept， Sloc）

SD

SD的码为Sno， DL的码为Sdept。

DL

三、3NF

关系模式$R<U，F>$中若不存在这样的码X、属性组Y及非主属性Z（$Z \not\subset Y$），使得$X \rightarrow Y$，$Y \not\rightarrow X$，$Y \rightarrow Z$，成立，则称$R<U，F> \in 3NF$。

例， SL(Sno, Sdept, Sloc) ∈ 2NF　SL(Sno, Sdept, Sloc) ∉ 3NF

SD（Sno， Sdept） ∈ 3NF　DL（Sdept， Sloc） ∈ 3NF

- 若R∈3NF，则R的每一个非主属性既不部分函数依赖于候选码也不传递函数依赖于候选码。
- 如果R∈3NF，则R也是2NF。
- 采用模式分解法将一个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，并不能完全消除关系模式中的各种异常情况和数据冗余。

四、 BC范式（BCNF）

设关系模式R<U，F>∈1NF，如果对于R的每个函数依赖X→Y，若Y不属于X，则X必含有候选码，那么R∈BCNF。

▪等价于：每一个决定属性因素都包含码

若R∈BCNF

❖R中的所有属性（主，非主）依赖于码

❖R∈3NF（证明）

若R∈3NF　则　R不一定∈B

▪证明：采用反证法。设R不是3NF。则必然存在如下条件的函数依赖，X→Y（Y⊄X），Y→Z，其中X是含有码的属性，Y是任意属性组，Z是非主属性，Z⊄Y，这样Y→Z函数依赖的决定因素Y不包含候选码，这与BCNF范式的定义相矛盾，所以如果R∈BCNF，则R也是3NF

例1 对关系模式C、SC、S进行分析。

C(Cno,Cname,Pcno)

SC(Sno,Cno,Grade)

S(Sno,Sname,Sdept,Sage)

▪它只有一个码Cno，这里没有任何属性对Cno部分依赖或传递依赖，所以C∈3NF。同时C中Cno是唯一的决定因素，所以C∈BCNF。

假定Sname也具有唯一性，那么S就有两个码，这两个码都由单个属性组成，彼此不相交。其他属性不存在对码的传递依赖与部分依赖，所以S∈3NF。同时S中除Sno，Sname外没有其他决定因素，所以S∈BCNF。

例2：在关系模式STC（S，T，C）中，S表示学生，T表示教师，C表示课程。

每一教师只教一门课。每门课由若干教师教，某一学生选定某门课，就确定了一个固定的教师。某个学生选修某个教师的课就确定了所选课的名称 。由语义可得到下面的函数依赖：

(S，C)→T，(S，T)→C，T→C

- 候选码为：
- (S，C)和(S，T)
- S、T、J都是主属性，所以STC∈3NF
- T→C，T是决定属性集，
- T不是候选码STC∈BCNF

例3：关系模式SCG(S,C,G)中，S是学生，C表示课程，G表示名次。

每一个学生选修每门课程的成绩有一定的名次，每门课程中每一名次只有一个学生（即没有并列名次）。由语义可得到下面的函数依赖:

(S，C)→G ；(C，G)→S

候选码为:(S，C)与(C，G)

- 这两个码各由两个属性组成，而且它们是相交的。这个关系模式中显然没有属性对码传递依赖或部分依赖。所以SCG∈3NF，而且除(S,C)与(C,G)以外没有其它决定因素，所以SCG∈BCNF

3NF与BCNF的关系与区别

- 如果关系模式R∈BCNF，必定有R∈3NF
- 如果R∈3NF，且R只有一个候选码，则R必属于BCNF。

- 一个模式中的关系模式如果都属于BCNF，那么在函数依赖范畴内，它已实现了彻底的分离，已消除了插入和删除的异常。
- 3NF的“不彻底”性表现在可能存在主属性对码的部分依赖和传递依赖。

规范化总结

- 规范化的目的：解决插入、删除、更新异常以及数据冗余度高的问题；
- 规范化的方法：通过模式分解，从模式中个属性间的函数依赖着手，尽量做到每个模式表示客观世界中的一个“事件”；

❖ 关系模式规范化的基本步骤

消除非码的决定因素

1NF

↓ 消除非主属性对码的部分函数依赖

2NF

↓ 消除非主属性对码的传递函数依赖

3NF

↓ 消除主属性对码的部分和传递函数依赖

BCNF

综合例子

例1：试问下列关系模式最高属第几范式，并解释其原因

1）R{(A,B,C,D) ,(B→ D,(A,B)→ C)}

方法：

第一步：确定候选码

(A,B)

第二步：判断是否满足BCNF(即判断决定因素是否含有码)；

第三步：判断非主属性是否满足完全函数依赖和直接函数依赖。

非主属性（C,D）, B→D,(A,B) →D)

所以：R ∈1NF

试问下列关系模式最高属第几范式，并解释其原因

1）R{(A,B,C,D) ,(A→ C, (C,D)→ B)}

2）R{(A,B,C,D) ,(A→ C, D → B)}

3）R{(A,B,C) ,(A → B, B→ A ,A → C}

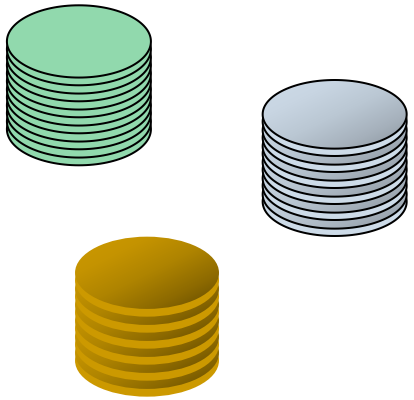
▪小结

了解数据不规范化带来的问题：
数据冗余、插入异常、删除异常、修改异常；

重点掌握几个概念：

- 函数依赖、非平凡和平凡函数依赖、部分和完全函数依赖、直接和传递函数依赖；
- 关系的候选码含义；
- 各种关系范式的含义：第一范式、第二范式、第三范式、BCNF

熟练掌握：给定某个关系模式能判别属于第几范式。



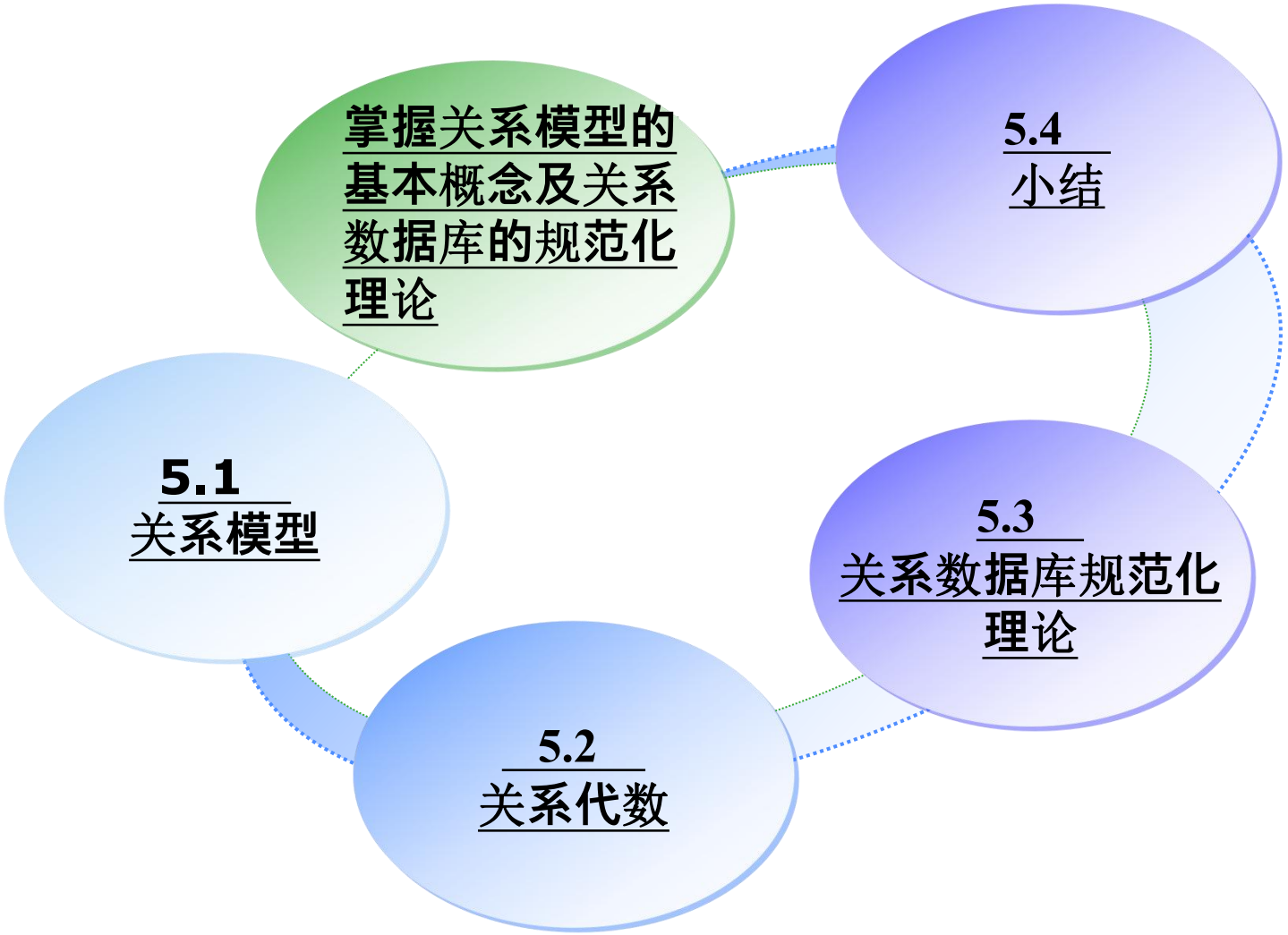
数据库系统

第五章 关系数据库基本理论

北京工业大学耿丹学院

计算机科学与技术专业

本章概述


掌握关系模型的基本概念及关系数据库的规范化理论
5.4
小结
5.1
关系模型
5.3
关系数据库规范化理论
5.2
关系代数

5.1 关系模型

- 关系数据结构
- 关系数据操作
- 关系的完整性约束

5.1.1 关系数据结构

- 关系模型建立在集合代数的基础上
- 关系数据结构的基本概念
  - 关系
  - 关系模式
  - 关系数据库

一、关系

1. 域（Domain）
2. 笛卡尔积（Cartesian Product）
3. 关系（Relation）

5.1.1 关系数据结构

一、关系

域（Domain）

- 一组值的集合，这组值具有相同数据类型。

例如：整数集合，实数集合，字符串集合，{男, 女} 等都是域。

- 基数：域中元素的个数称为域的基数。

D_1={教授, 副教授, 讲师, 助教}，表示职称的集合；
其中D_1的基数是4

5.1.1 关系数据结构

一、关系

笛卡尔积（Cartesian Product）

1） 笛卡尔积的定义

给定一组域D_1，D_2，…，D_n，这些域中可以有相同的。D_1，D_2，…，D_n的笛卡尔积为：

$$D_1 \times D_2 \times \ldots \times D_n = \{ (d_1, d_2, \ldots, d_n) \mid d_i \in D_i, \ i=1, 2, \ldots, n \}$$

- 所有域的所有取值的一个组合
- 不能重复

5.1.1 关系数据结构

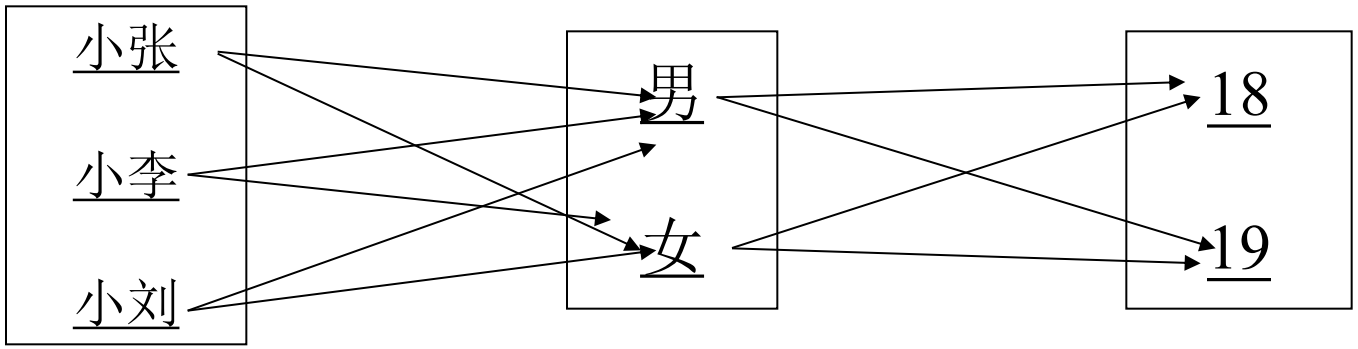
例：

D_1= 姓名 = { 小张，小李，小刘 }

D_2= 性别 = {男，女}

D_3= 年龄 = {18，19}

求D_1，D_2，D_3的笛卡尔积？



$D1 \times D2 \times D3$ =

{(小张，男，18)，(小张，男，19)，(小张，女，18)，(小张，女，19)，
(小李，男，18)，(小李，男，19)，(小李，女，18)，(小李，女，19)，
(小刘，男，18)，(小刘，男，19)，(小刘，女，18)，(小刘，女，19)}

笛卡尔积（Cartesian Product）

2) 元组（Tuple）

笛卡尔积中每一个元素（d_1，d_2，…，d_n）叫作一个n元组（n-tuple）或简称元组。

3) 分量（Component）

笛卡尔积元素（d_1，d_2，…，d_n）中的每一个值d_i叫作一个分量。

4) 基数（Cardinal number）

若D_i（i=1，2，…，n）为有限集，其基数为m_i（i=1，2，…，n），则$D_1 \times D_2 \times \ldots \times D_n$的基数M为：

$$M = \prod_{i=1}^{n} m_i$$

笛卡尔积（Cartesian Product）

5)笛卡尔积的表示方法

笛卡尔积可表示为一个二维表。表中的每行对应一个元组，表中的每列对应一个域。

在上述例中，12个元组可列成一张二维表

表1：

姓名	性别	年龄
小张	男	18
小张	男	19
小张	女	18
小张	女	19
小李	男	18
小李	男	19
小李	女	18
小李	女	19
小刘	男	18
小刘	男	19
小刘	女	18
小刘	女	19

5.1.1 关系数据结构

关系（Relation）

1) 关系的定义

笛卡尔积$D_1 \times D_2 \times \ldots \times D_n$的子集叫作在域$D_1$，$D_2$，…，$D_n$上的关系。可表示为:

$R（D_1，D_2，\ldots，D_n）$

***R*：关系名**

***n*：关系的目或度（Degree）**

- 关系是笛卡尔积的有限子集。
- 关系是一个二维表。

5.1.1 关系数据结构

在表1的笛卡尔积中取出有意义的元组来构造一个学生关系：

学生(姓名，性别，年龄)

- 假设小张和小李是男生且均为18岁，小刘是女生，19岁。

表2：学生关系

姓名	性别	年龄
小张	男	18
小李	男	18
小刘	女	19

表1： D_1，D_2，D_3的笛卡尔积

姓名	性别	年龄
小张	男	18
小张	男	19
小张	女	18
小张	女	19
小李	男	18
小李	男	19
小李	女	18
小李	女	19
小刘	男	18
小刘	男	19
小刘	女	18
小刘	女	19

5.1.1 关系数据结构

关系（Relation）

2) 元组

- 关系中的每一行称作一个元组
- 组成元组的元素为分量。
- 如表2中有三个元组

表2：学生关系

姓名	性别	年龄
小张	男	18
小李	男	18
小刘	女	19

3) 属性

- 关系中的每一列称为一个属性。
- 如表2中有三个属性，分别为：姓名、性别和年龄。
- 关系中的属性名具有标识列的作用，则同一关系中的属性名（列名）不能相同。

5.1.1 关系数据结构

关系（Relation）

4) 码

- 码：在关系中唯一标识元组的最小的属性集称为该关系的码或关键字。

- 候选码：一个关系中可能有若干个码，它们称为该关系的候选码或候选关键字。

  - ✓ 任何候选码中的属性为主属性；
  - ✓ 不包含在候选码中的属性为非主属性

- 主码：当一个关系有多个候选码时，应选定其中的一个候选码为主码。一般主码也简称码。

学生关系

学号	姓名	性别	年龄
05801	小张	男	18
05802	小李	男	18
05803	小刘	女	19

学生选课关系

学号	课程名	课时
05801	C	48
05801	数据库	56
05802	C	48

5.1.1 关系数据结构

- 外码：如果关系A中的某属性集是关系B的码，但不是A的码，则称该属性集为A的外码或外关键字。

A：学生登记表

学号	姓名	学院编号
05801	小张	01
05802	小李	02
05803	小王	02

B: 学院登记表

学院编号	学院名称
01	信息
02	经管
03	自动化

关系A*称为*参照关系　　关系B*称为*被参照关系或目标关系。

说明：

- 关系*A*和*B*不一定是不同的关系。
- 目标关系*B*的主码和参照关系A的外码必须定义在同一个（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，往往取相同的名字，以便于识别。

5.1.1 关系数据结构

- 全码：

✓ 若关系中的候选码只包含一个属性，则称它为单属性码。

✓ 若候选码是由多个属性构成，则称它为多属性码。

✓ 若关系中只有一个候选码，且这个候选码中包括全部属性，则该候选码称为全码。

学生选课关系

学号	课程名
05801	C
05801	数据库
05802	C

5.1.1 关系数据结构

6) 数据库中基本关系的性质

① 同一属性的数据具有同质性

✓ 每一列中的分量是同一类型的数据，来自同一个域。

✓ 例如：学生选课表的结构中：选课（学号，课号，成绩），而成绩的属性值不能有百分制、5分制、或“优”“良”等多种取值法。

② 同一关系的属性名具有不能重复性

✓ 同一关系中不同属性的数据可出自同一个域，但是不同的属性要给予不同的属性名。

✓ 如：要设计一个能存储两科成绩的学生成绩表，其表结构不能写为：学生成绩（学号，成绩，成绩）

可以设计为：学生成绩（学号，成绩1，成绩2）

③ 关系中的列位置具有顺序无关性

✓ 关系中列的次序可以任意交换、重新组织，属性顺序不影响使用。

5.1.1 关系数据结构

④ 关系中的元组具有无冗余性

✓ 关系中的任意两个元组不能完全相同。

⑤ 关系中的元组位置具有顺序无关性

✓ 关系元组的顺序可以任意交换。

⑥ 关系中每一个分量都必须是不可分的数据项

✓ 关系规范条件中最基本的一条就是关系的每一个分量必须是不可分的数据项，即分量是原子量。

姓名	所在系	成绩	
		C成绩	数据结构成绩
刘克	计算机	89	90
李明	经管	86	88

姓名	所在系	C成绩	数据结构成绩
刘克	计算机	89	90
李明	经管	86	88

二、关系模式

关系模式的定义

- 关系的描述称为关系模式。关系模式可以形式化地表示为：

$$R（U，D，\mathrm{dom}，F）$$

其中：

R：关系名；

U：组成该关系的属性名集合；

D：属性集*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

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。

二、关系模式

关系模式的定义

- 关系模式通常可以简记为

$R(U)$ 或 $R(A_1, A_2, \ldots An)$

其中：R 关系名

$A_1, A_2, \ldots, A_n$ 属性名

U 是属性的集合

$U=\{A_1, A_2, \ldots, A_n\}$

二、关系模式

关系模式与关系

- 关系模式是对关系的描述，是关系的型，即框架或结构。是静态的，稳定的。
- 关系是按关系模式组织的表格。是关系模式在某一时刻的状态或内容。是动态的。

关系模式和关系往往统称为关系，通过上下文加以区别

三、关系数据库

关系数据库的定义

在一个给定的应用领域中，所有实体及实体之间联系所形成的关系的集合构成一个关系数据库。

关系数据库的型和值

关系数据库的型称为关系数据库模式，是对关系数据库的描述。

关系数据库的值是这些关系模式在某一时刻对应的关系的集合，也就是所说的关系数据库的数据。

5.1.2 关系操作

一、基本的关系操作

- 查询

  选择、投影、连接、除、并、交、差

- 更新

  插入、删除、修改

- 查询的表达能力是其中最主要的部分

关系操作的特点：集合操作方式

即操作的对象和结果都集合。

5.1.2 关系操作

二、关系数据语言的分类

- 关系代数语言
  - 对关系的运算是用代数方式来表达查询要求
- 关系演算语言：用谓词演算（逻辑方式）来表达查询要求的查询语言

- 上述两种的特点：
  - 抽象
  - 常用评估实际系统查询语言能力的标准或理论基础

- 具有关系代数和关系演算双重特点的语言
  - 典型代表：**SQL**（结构化查询语言）

5.1.3 关系的完整性

一、关系的三类完整性约束

关系模型的完整性规则是对关系的某种约束条件。

关系模型中三类完整性约束：

实体完整性

参照完整性

用户定义的完整性

实体完整性和参照完整性是关系模型必须满足的完整性约束条件，被称作是关系的两个不变性，应该由关系系统自动支持。

用户定义的完整性是应用领域需要遵循的约束条件。

5.1.3 关系的完整性

二、 实体完整性

1、实体完整性规则（Entity Integrity）：

关系数据库中所有的表都必须有主码，要求主码不能为空，且不能取相同的值。

例： 学生(学号，姓名，性别，年龄)

学号属性为主码，则其不能取空值

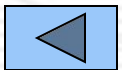
2、说明：

例如：

实体完整性规则规定基本关系的主码不能取空值。

例：选修（学号，课程号，成绩）

“学号、课程号”为主码，则两个属性都不能取空值。



5.1.3 关系的完整性

三、 参照完整性

1. 关系间的引用

学生

学号	姓名	性别	专业号	年龄
801	张三	女	01	19
802	李四	男	01	20
803	王五	男	01	20
804	赵六	女	02	20
805	钱七	男	02	19

在关系模型中实体及实体间的联系都是
存在着关系与关系间的引用。

例1 学生实体、专业实体以及专业与学生间的一对多联系。

学生（学号，姓名，性别，专业号，年龄）

专业（专业号，专业名）

专业

专业号	专业名
01	信息
02	数学
03	计算机

学生关系中每个元组的“专业号”属性只取下面两类值：

（1）空值，表示尚未给该学生分配专业

（2）非空值，这时该值必须是专业关系中某个元组的“专业号”值，表示该学生不可能分配到一个不存在的专业中。

5.1.3 关系的完整性

例2 学生、课程、学生与课程之间的多对 多

学生（学号，姓名，性别，专业号，年龄）

课程（课程号，课程名，学分）

选修（学号，课程号，成绩）

选修（学号，课程号，成绩）

“学号”和“课程号”是选修关系中的主属性按照实体完整性和参照完整性规则，它们只能取相应被参照关系中已经存在的主码值。

学生

学号	姓名	性别	专业号	年龄
801	张三	女	01	19
802	李四	男	01	20
803	王五	男	01	20
804	赵六	女	02	20
805	钱七	男	02	19

课程

课程号	课程名	学分
01	数据库	4
02	数据结构	4
03	编译	4
04	PASCAL	2

学生选课

学号	课程号	成绩
801	04	92
801	03	78
801	02	85
802	03	82
802	04	90
803	04	88

5.1.3 关系的完整性

例3 学生实体及其内部的领导联系(一对多)

学生（学号，姓名，性别，专业号，年龄，班长）

学生

学号	姓名	性别	专业号	年龄	班长
801	张三	女	01	19	802
802	李四	男	01	20	802
803	王五	男	01	20	802
804	赵六	女	02	20	805
805	钱七	男	02	19	805

学生（学号，姓名，性别，专业号，年龄，班长）

“班长”属性值可以取两类值：

（1）空值，表示该学生所在班级尚未选出班长，或该学生本人即是班长；

（2）非空值，这时该值必须是本关系中某个元组的学号值。

5.1.3 关系的完整性

2. 参照完整性规则

若属性（或属性组）***F***是关系***R***的外码，它与关系***S***的主码相对应（关系***R***和***S***不一定是不同的关系），则对于***R***中每个元组在***F***上的值必须为：

- 或者取空值（***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。

5.1.3 关系的完整性

四、 用户定义的完整性

用户定义的完整性是应用领域需要遵循的约束条件，是针对不同的应用环境而定义的约束条件，反映某一具体应用所涉及的数据应该满足的语义要求。

例：课程(课程号，课程名，学分)

- “课程号”属性必须取唯一值
- 非主属性“课程名”也不能取空值
- “学分”属性只能取值{1，2，3，4，5}

5.2 关系代数

一、概述

1. 关系代数

一种抽象的查询语言对关系的运算来表达查询

2. 运算的三要素：　运算对象、运算结果、运算符

3. 关系代数运算的三个要素

运算对象：关系
运算结果：关系
运算符：四类

4. 关系代数运算的分类

- 传统的集合运算：并、差、交、广义笛卡尔积
- 专门的关系运算：选择、投影、连接、除

关系代数运算符

1. **集合运算符**
   - 将关系看成元组的集合
   - 运算是从关系的“水平”方向即行的角度来进行
2. **专门的关系运算符**

   不仅涉及行而且涉及列
3. **算术比较符**

   辅助专门的关系运算符进行操作
4. **逻辑运算符**

   辅助专门的关系运算符进行操作

5.2 关系代数

表1 关系代数运算符

运算符		含义	运算符		含义
集合运算符	∪	并	比较运算符	>	大于
	-	差		≥	大于等于
	∩	交		<	小于
	×	广义笛卡尔积		≤	小于等于
				= ≠	等于 不等于
专门的关系运算符	σ	选择	逻辑运算符	¬	非
	π	投影		∧	与
	⋈	连接		∨	或
	÷	除			

二、传统的集合运算

- 并　$R \cup S = \{ t \mid t \in R \vee t \in S \}$
- 差　$R - S = \{ t \mid t \in R \wedge t \notin S \}$
- 交　$R \cap S = \{ t \mid t \in R \wedge t \in S \}$　或　$R \cap S = R - (R - S)$

要求R和S

- 具有相同的目n（即两个关系都有n个属性）
- 相应的属性取自同一个域

- 广义笛卡尔积

$$R \times S = \{ \widehat{t_r\ t_s} \mid t_r \in R \wedge t_s \in S \}$$

R (n目关系，k_1个元组)　　S (*m*目关系，k_2个元组)

R×*S*

列：(n+m)列的元组的集合；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：$k_1 \times k_2$个元组

5.2 关系代数

R

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

S

A	B	C
a1	b2	c2
a1	b3	c2
a2	b2	c1

$R \cup S$

A	B	C
a1	b1	c1
a1	b2	c2
a1	b3	c2
a2	b2	c1

R-S

A	B	C
a1	b1	c1

$R \cap S$

A	B	C
a1	b2	c2
a2	b2	c1

$R \times S$

R.A	R.B	R.C	S.A	S.B	S.C
a1	b1	c1	a1	b2	c2
a1	b1	c1	a1	b3	c2
a1	b1	c1	a2	b2	c1
a1	b2	c2	a1	b2	c2
a1	b2	c2	a1	b3	c2
a1	b2	c2	a2	b2	c1
a2	b2	c1	a1	b2	c2
a2	b2	c1	a1	b3	c2
a2	b2	c1	a2	b2	c1

三、 专门的关系运算

- 投影
- 选择
- 连接
- 除

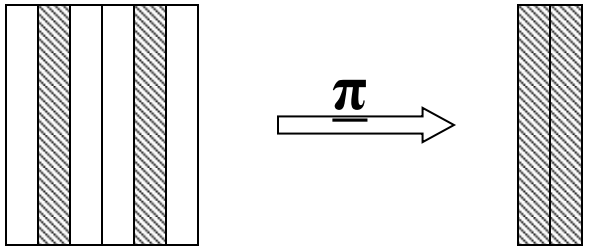
1. 投影（Projection）

1）投影运算符的含义

从*R*中选择出若干属性列组成新的关系

$\pi_A(R) = \{ t[A] \mid t \in R \}$

A：*R*中的属性列



2）投影操作主要是从列的角度进行运算

注：但投影之后不仅取消了原关系中的某些列，而且还可能取消某些元组（避免重复行）

Student

学 号 Sno	姓 名 Sname	性 别 Ssex	年 龄 Sage	所在系 Sdept
01	李勇	男	20	CS
02	刘晨	女	19	IS
03	王敏	女	18	MA
04	张立	男	19	IS

5.2 关系代数

3）举例

Student

学 号 Sno	姓 名 Sname	性 别 Ssex	年 龄 Sage	所在系 Sdept
01	李勇	男	20	CS
02	刘晨	女	19	IS
03	王敏	女	18	MA
04	张立	男	19	IS

[例1] 查询学生的姓名和所在系

即求Student关系上学生姓名和所在系两个属性上的投影

$\Pi_{Sname,\ Sdept}(Student)$

或 $\Pi_{2,\ 5}(Student)$

结果：

Sname	Sdept
李勇	CS
刘晨	IS
王敏	MA
张立	IS

[例2] 查询学生关系Student中都有哪些系

$\pi_{Sdept}(Student)$

结果：

Sdept
CS
IS
MA

2. 选择（Selection）

1) 选择又称为限制（Restriction）

2) 选择运算符的含义

在关系***R***中选择满足给定条件的诸元组。

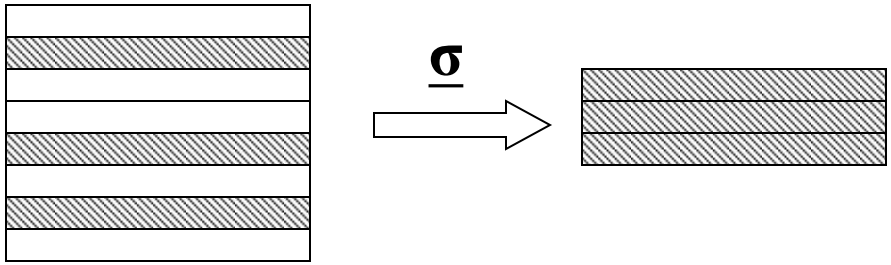
$$\sigma_F(R) = \{t \mid t \in R \wedge F(t) = \text{'真'}\}$$

σ为选择运算符

F：选择条件，是一个逻辑表达式，由运算对象（属性名（序号）、常数、简单函数）、算术运算符和逻辑运算符连接。

2. 选择（Selection）

3) 选择运算是从行的角度进行的运算



4) 举例

5.2 关系代数

4)举例：设有一个学生-课程数据库，包括学生关系Student、课程关系Course和选修关系SC。

Student

学 号 Sno	姓 名 Sname	性 别 Ssex	年 龄 Sage	所在系 Sdept
01	李勇	男	20	CS
02	刘晨	女	19	IS
03	王敏	女	18	MA
04	张立	男	19	IS

Course

课程号 Cno	课程名 Cname	先行课 Cpno	学分 Ccredit
1	数据库	5	4
2	数学		2
3	信息系统	1	4
4	操作系统	6	3
5	数据结构	7	4
6	数据处理		2
7	C语言	6	4

SC

学 号 Sno	课程号 Cno	成 绩 Grade
01	1	92
01	2	85
01	3	88
02	2	90
02	3	80

[例3] 查询信息系（IS系）全体学生学号和姓名

[例4] 查询年龄小于20岁的学生的姓名

5.2 关系代数

[例3] 查询信息系（IS系）全体学生的学号和姓名

$\Pi_{sno,\ sname}(\sigma_{Sdept\ =\ 'IS'}(Student))$

或 $\Pi_{1,\ 5}(\sigma_{5\ =\ 'IS'}(Student))$

Sno	Sname	Ssex	Sage	Sdept
02	刘晨	女	19	IS
04	张立	男	19	IS

[例4] 查询年龄小于20岁的学生的姓名

$\Pi_{sname}(\sigma_{Sage < 20}(Student))$

或

$\Pi_{2}(\sigma_{4 < 20}(Student))$

Sno	Sname	Ssex	Sage	Sdept
02	刘晨	女	19	IS
03	王敏	女	18	MA
04	张立	男	19	IS

3. 连接（Join）

1）连接也称为θ连接

2）连接运算的含义

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

$$R \underset{A\theta B}{\bowtie} S = \{ \widehat{t_r t_s} \mid t_r \in R \wedge t_s \in S \wedge t_r[A]\theta t_s[B] \}$$

A和B：分别为R和S上度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$R \times S$中选取（R关系）在A属性组上的值与（S关系）在B属性组上值满足比较关系的元组。

5.2 关系代数

[例5]

R

A	B	C
a_1	b_1	5
a_1	b_2	6
a_2	b_3	8
a_2	b_4	12

S

B	E
b_1	3
b_2	7
b_3	10
b_3	2
b_5	2

$R \times S$

A	R.B	C	S.B	E
a_1	b_1	5	b_1	3
a_1	b_1	5	b_2	7
a_1	b_1	5	b_3	10
a_1	b_1	5	b_3	2
a_1	b_1	5	b_5	2
a_1	b_2	6	b_1	3
a_1	b_2	6	b_2	7
a_1	b_2	6	b_3	10
a_1	b_2	6	b_3	2
a_1	b_2	6	b_5	2
a_2	b_3	8	b_1	3
…	…	…	…	…

$R \underset{C<E}{\bowtie} S$

A	R.B	C	S.B	E
a_1	b_1	5	b_2	7
a_1	b_1	5	b_3	10
a_1	b_2	6	b_2	7
a_1	b_2	6	b_3	10
a_2	b_3	8	b_3	10

3. 连接（Join）

3）两类常用连接运算

等值连接（equijoin）

θ为“＝”的连接运算称为等值连接

等值连接的含义：

从关系R与S的广义笛卡尔积中选取A、B属性值相等的那些元组，即等值连接为：

$$R \underset{A=B}{\bowtie} S = \{ t_r \widehat{\ } t_s \mid t_r \in R \wedge t_s \in S \wedge t_r[A] = t_s[B] \}$$

自然连接（Natural join）

自然连接是一种特殊的等值连接

两个关系中进行比较的分量必须是相同的属性组,在结果中把重复的属性列去掉。

自然连接的含义：（R和S具有相同的属性组B）

$$R \bowtie S = \{ t_r \widehat{\ } t_s \mid t_r \in R \wedge t_s \in S \wedge t_r[B] = t_s[B] \}$$

5.2 关系代数

[例6]

R

A	B	C
a_1	b_1	5
a_1	b_2	6
a_2	b_3	8
a_2	b_4	12

S

B	E
b_1	3
b_2	7
b_3	10
b_3	2
b_5	2

$R \times S$

A	R.B	C	S.B	E
a_1	b_1	5	b_1	3
a_1	b_1	5	b_2	7
a_1	b_1	5	b_3	10
a_1	b_1	5	b_3	2
a_1	b_1	5	b_5	2
a_1	b_2	6	b_1	3
a_1	b_2	6	b_2	7
a_1	b_2	6	b_3	10
a_1	b_2	6	b_3	2
a_1	b_2	6	b_5	2
a_2	b_3	8	b_1	3
…	…	…	…	…

等值连接 $R \bowtie S$ $R.B=S.B$

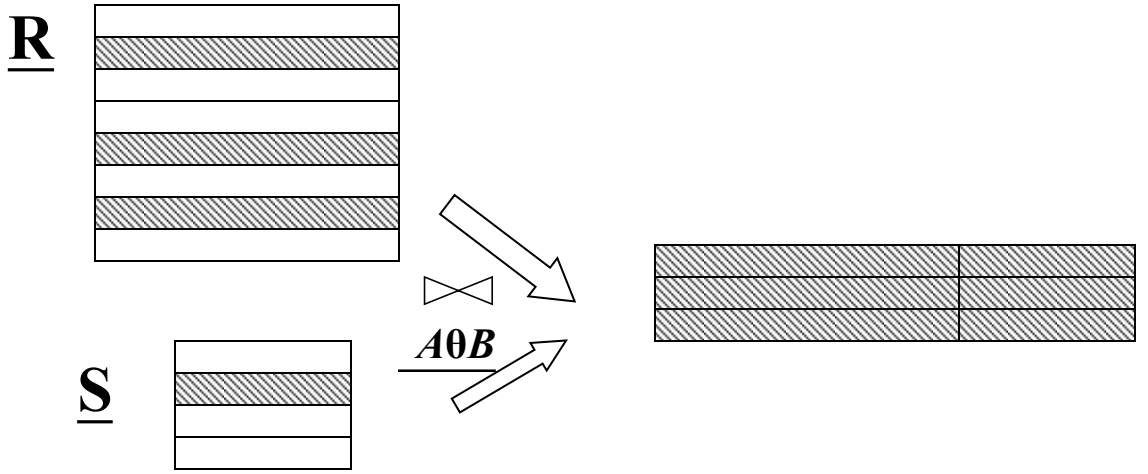
A	R.B	C	S.B	E
a_1	b_1	5	b_1	3
a_1	b_2	6	b_2	7
a_2	b_3	8	b_3	10
a_2	b_3	8	b_3	2

自然连接 $R \bowtie S$

A	B	C	E
a_1	b_1	5	3
a_1	b_2	6	7
a_2	b_3	8	10
a_2	b_3	8	2

3. 连接（Join）

4）一般的连接操作是从行的角度进行运算。



自然连接还需要取消重复列，所以是同时从行和列的角度进行运算。

外连接

- 外连接(Outer join): 在自然连接中把舍弃的元组也保存在结果关系中,其它属性上填空值(Null)
- 左连接(Left outer join): 只保留左边关系要舍去的元组
- 右连接(Right outer join): 只保留右边关系要舍去的元组

A	B	C	E
a_1	b_1	5	3
a_1	b_2	6	7
a_2	b_3	8	10
a_2	b_3	8	2
a_2	b_4	12	Null
Null	b_5	Null	2

(a) R、S外连接

A	B	C	E
a_1	b_1	5	3
a_1	b_2	6	7
a_2	b_3	8	10
a_2	b_3	8	2
a_2	b_4	12	Null

(b) R、S左连接

A	B	C	E
a_1	b_1	5	3
a_1	b_2	6	7
a_2	b_3	8	10
a_2	b_3	8	2
Null	b_5	Null	2

(c) R、S右连接

4. 综合举例

以学生-课程数据库为例,数据库中包含3个关系:

S(Sno, Sname, Sage,Ssex,Sdept)

SC(Sno,Cno,Grade);

C(Cno,Cname,Cpno,Ccredit)

[例1] 检索学习课程号为2的学生学号和成绩

$$\pi_{Sno，Grade}(\sigma_{Cno='2'}(SC))$$

或$\pi_{1，3}(\sigma_{2='2'}(SC))$

[例2] 检索学习课程号为2的学生学号和姓名

$$\pi_{Sno，Sname}(\sigma_{Cno='2'}(S \bowtie SC))$$

[例 3] 检索课程名为数据结构的学生学号与姓名

$$\pi_{Sno,Sname}(\sigma_{Cname='数据结构'}(C \bowtie SC \bowtie S))$$

[例4] 检索选修课程号为2或4的学生学号

$$\Pi_{Sno}(\sigma_{Cno='2' \vee Cno='4'}(SC))$$

[例5] 检索不选修课程号为2的学生姓名与年龄

$$\pi_{Sname,\ Sage}(S) - \pi_{Sname,\ Sage}(\sigma_{Cno='2'}(S \bowtie SC))$$

关系模型和关系代数小结

- 理解关系、关系模式、关系数据库的形式化定义
- 掌握码的相关概念：码，候选码、主码、外码、全码
- 掌握关系的三类完整性约束
- 掌握关系代数的常见运算，能熟练使用关系代数表达式来表达查询。

5.3.1 问题的提出

一、概念回顾
二、关系模式的形式化定义
三、什么是数据依赖
四、关系模式的简化定义
五、数据依赖对关系模式影响

一、概念回顾

- 关系：

  描述实体、属性、实体间的联系。

  - 从形式上看，它是一张二维表，是所涉及属性的笛卡尔积的一个子集。

- 关系模式：
- 用来定义关系。
- 关系数据库：
- 基于关系模型的数据库，利用关系来描述现实世界。
  - 从形式上看，它由一组关系组成。
- 关系数据库的模式：
- 定义这组关系的关系模式的全体。

二、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

关系模式的简化表示

关系模式R（U, D, DOM, F）

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

三、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

- 是通过一个关系中属性间值的相等与否体现出来的数据间的相互关系；
- 是现实世界属性间相互联系的抽象，是数据内在的性质，是语义的体现。

3. 数据依赖的主要类型

- 函数依赖（**Functional Dependency**，简记为**FD**）
- 多值依赖（**Multivalued Dependency**，简记为**MVD**）

四、数据依赖对关系模式的影响

例：描述学校数据库：

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、F>**

U ＝｛ Sno, Sdept, Mname, Cname, Grade ｝

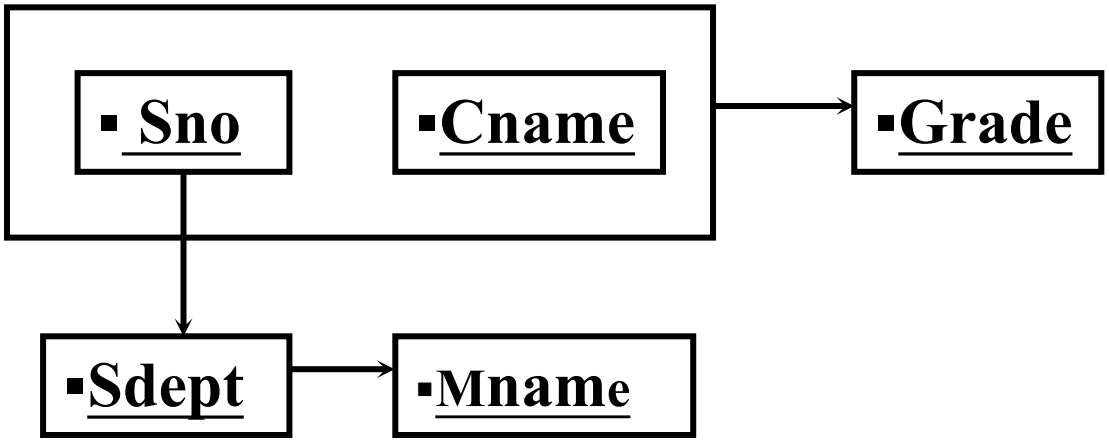
学校数据库的语义：

1. 一个系有若干学生， 一个学生只属于一个系；
2. 一个系只有一名主任；
3. 一个学生可以选修多门课程， 每门课程有若干学生选修；
4. 每个学生所学的每门课程都有一个成绩。

五、数据依赖对关系模式的影响（续）

属性组U上的一组函数依赖F：

$$F = \{ Sno \rightarrow Sdept,\ Sdept \rightarrow Mname,\ (Sno, Cname) \rightarrow Grade \}$$



▪某一时刻关系模式Student<U,F>的一个实例：

Sno	Sdept	Mname	Cname	Grade
S1	信息系	王主任	数据库	90
S2	信息系	王主任	数据库	87
S3	信息系	王主任	数据库	99
S4	信息系	王主任	数据库	80
S5	信息系	王主任	数据库	88
…	…	…	…	…

▪存在的问题：

- 1. 数据冗余太大——浪费大量的存储空间

2. 更新异常（Update Anomalies）

数据冗余，更新数据时，维护数据完整性代价大。

▪例：每一个系主任的姓名重复出现

例：某系更换系主任后，系统必须修改与该系学生有关的每一个元组。

- 某一时刻关系模式Student<U,F>的一个实例：

Sno	Sdept	Mname	Cname	Grade
S1	信息系	王主任	数据库	90
S2	信息系	王主任	数据库	87
S3	信息系	王主任	数据库	99
S4	信息系	王主任	数据库	80
S5	信息系	王主任	数据库	88
…	…	…	…	…

- 存在的问题：

3. 插入异常（Insertion Anomalies）

该插的数据插不进去

例，如果一个系刚成立，尚无学生，我们就无法把这个系及其系主任的信息存入数据库。

4. 删除异常（Deletion Anomalies）

不该删除的数据不得不删

例，如果某个系的学生全部毕业了，我们在删除该系学生信息的同时，把这个系及其系主任的信息也丢掉了。

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。

- 分解成三个关系模式：
- S (Sno, Sdept, Sno → Sdept);
- SC (Sno, Cname, Grade, (Sno, Cname) → Grade);
- DEPT (Sdept, Mname, Sdept→Mname);

5.3.2 规范化

规范化理论正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、删除异常、更新异常和数据冗余问题。

5.3.2 规范化

- 函数依赖
- 范式
- 规范化总结
- 例子

一、函数依赖

定义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。

X称为这个函数依赖的决定属性集(决定因素)。　　Y=f(x)

说明：1. 函数依赖指R的所有关系实例均要满足的约束条件。

2. 函数依赖是语义范畴的概念。只能根据数据的语义来确定函数依赖。例如“姓名→年龄”函数依赖只在不允许有同名的条件下成立

3. 数据库设计者可以对现实世界作强制的规定。例如规定不允许同名人出现，函数依赖“姓名→年龄”成立。所插入的元组必须满足规定的函数依赖，若有同名，则拒绝插入该元组。

例: Student(Sno, Sname, Ssex, Sage, Sdept)

假设不允许重名，则有:

Sno → Ssex，　Sno → Sage , Sno → Sdept，

Sno ←→ Sname, Sname → Ssex，　Sname → Sage

Sname → Sdept

但Ssex $\not\rightarrow$ Sage

若X→Y，并且Y→X，则记为X←→Y。

若Y不函数依赖于X，则记为X$\not\rightarrow$Y。

二、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 ⊆ X，则称X→Y是平凡的函数依赖

若X→Y，但Y ⊈ X，则称X→Y是非平凡的函数依赖

对于任一关系模式，平凡函数依赖都是必然成立的，它不反映新的语义，若不特别声明，总是讨论非平凡函数依赖。

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 (Sno, Cno) → Cno

三、完全函数依赖与部分函数依赖

定义2 在关系模式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完全函数依赖于X，记作 $X \xrightarrow{F} Y$。

若X→Y，但Y不完全函数依赖于X，则称Y部分函数依赖于X，记作 $X \xrightarrow{P} Y$。

例：在关系SC(Sno, Cno, Grade)中，

由于：Sno →Grade，Cno → Grade，

因此：(Sno, Cno) $\xrightarrow{F}$ Grade

- 而在Student(Sno,Sdept,Cno,Mname,Grade)中
- (Sno,Cno)→Sdept
- 是部分函数依赖
- 因为Sno →Sdept成立，且Sno是（Sno，Cno）的真子集

四、传递函数依赖

定义3　在关系模式R(U)中，如果X→Y，Y→Z，且 $Y \subseteq X$，$Y \not\to X$，则称Z传递函数依赖于X。

记为：$X \xrightarrow{传递} Z$

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

五、码

定义 4 设K为关系模式R<U,F>中的属性或属性组合。若 $K \xrightarrow{F} U$，则K称为R的一个侯选码（Candidate Key）。若关系模式R有多个候选码，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；
  - 不包含在任何码中的属性称为非主属性
- 全码 (ALL KEY)：整个属性组是码

[例]

关系模式S(Sno,Sdept,Sage)，单个属性Sno是码，

SC（Sno，Cno，Grade）中，（Sno，Cno）是码

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：

$$1NF \supset 2NF \supset 3NF \supset BCNF \supset 4NF \supset 5NF$$

- 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

一、1NF

- 定义

  如果一个关系模式R的所有属性都是不可分的基本数据项，则R∈1NF。

- 第一范式是对关系模式的最起码的要求。不满足第一范式的数据库模式不能称为关系数据库。
- 但是满足第一范式的关系模式并不一定是一个好的关系模式。

例: 关系模式 SLC(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。SLC的码为 (Sno, Cno)

函数依赖包括：

(Sno, Cno) $\xrightarrow{F}$ Grade

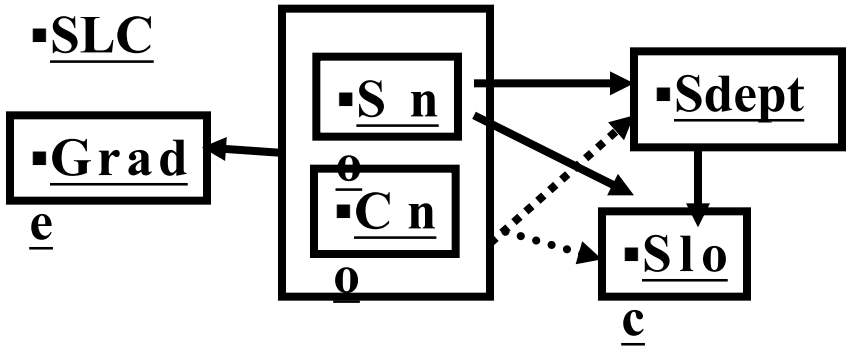
Sno → Sdept

(Sno, Cno) $\xrightarrow{P}$ Sdept

Sno → Sloc

(Sno, Cno) $\xrightarrow{P}$ Sloc

Sdept → Sloc



SLC满足第一范式。

非主属性Sdept和Sloc部分函数依赖于码(Sno, Cno)

SLC (Sno, Sdept, Sloc, Cno, Grade)不是一好的关系模式

(1) 插入异常

假设**Sno＝02，Sdept＝IS，Sloc＝N**的学生还未选课，因课程号是主属性，因此该学生的信息无法插入**SLC**。

(2) 删除异常

假定某个学生本来只选修了**3**号课程这一门课。现在因身体不适，他连**3**号课程也不选修了。因课程号是主属性，此操作将导致该学生信息的整个元组都要删除。

(3) 数据冗余度大

如果一个学生选修了**10**门课程，那么他的**Sdept**和**Sloc**值就要重复存储了**10**次。

(4) 修改复杂

例如学生转系，在修改此学生元组的**Sdept**值的同时，还可能需要修改住处（**Sloc**）。如果这个学生选修了**n**门课，则必须无遗漏地修改**n**个元组中全部**Sdept**、**Sloc**信息。

❖ 原因

Sdept、 Sloc部分函数依赖于码。

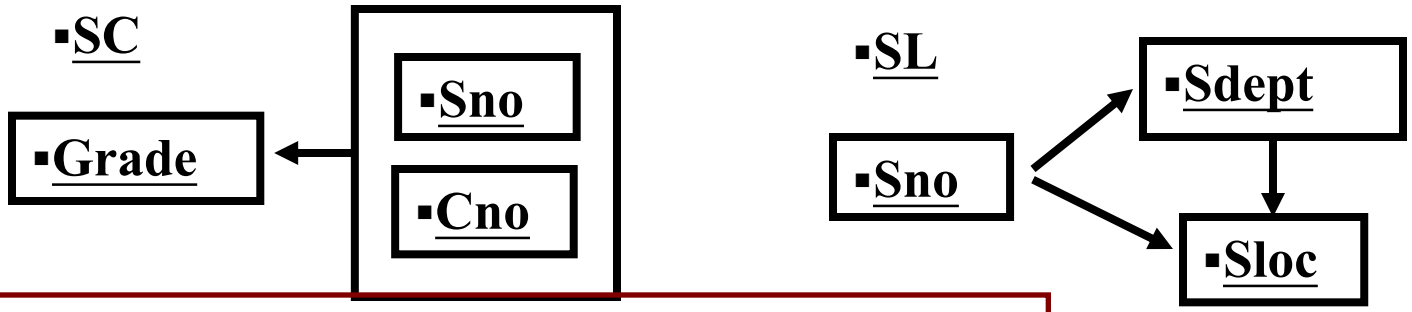
❖ 解决方法

SLC分解为两个关系模式，以消除这些部分函数依赖

SC（Sno， Cno， Grade）

SL（Sno， Sdept， Sloc）

❖ 函数依赖图：



❖关系模式SC的码为（Sno，Cno）

❖ 关系模式SL的码为Sno

❖这样非主属性对码都是完全函数依赖

二、2NF

定义 若关系模式R∈1NF，并且每一个非主属性都完全函数依赖于R的码，则R∈2NF。

例：SLC(Sno, Sdept, Sloc, Cno, Grade) ∈1NF

SLC(Sno, Sdept, Sloc, Cno, Grade) ∉ 2NF

SC（Sno， Cno， Grade） ∈ 2NF

SL（Sno， Sdept， Sloc） ∈ 2NF

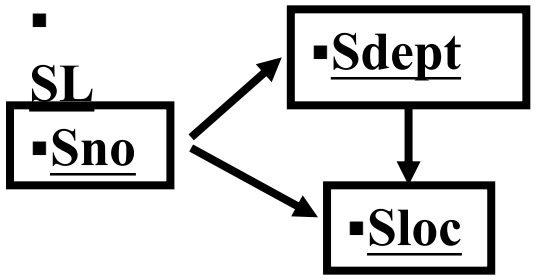
- 采用模式分解法将一个1NF的关系分解为多个2NF的关系，可以在一定程度上减轻原1NF关系中存在的插入异常、删除异常、数据冗余度大、修改复杂等问题。
- 将一个1NF关系分解为多个2NF的关系，并不能完全消除关系模式中的各种异常情况和数据冗余。

- 模式分解：就是将一个关系模式分解为若干个模式，分解后的模式具有下面的三个特征：
  - 分解后的模式均为高一级的模式
  - 分解后关系中的数据不会丢失，即分解后的关系再经连接后能恢复到原来的关系，称为无损连接
  - 分解后关系中的函数依赖不会丢失，这叫做依赖保持。

例：2NF关系模式SL(Sno, Sdept, Sloc)中

函数依赖： Sno→Sdept Sno→Sloc，
Sdept→Sloc

Sloc传递函数依赖于Sno，即SL中存在非主属性对码的传递函数依赖。



解决方法：

采用模式分解法，把SL分解为两个关系模式，以消除传递函数依赖：

- SD（Sno， Sdept）
- DL（Sdept， Sloc）

SD

SD的码为Sno， DL的码为Sdept。

DL

三、第三范式 （3NF）

关系模式*R<U，F>* ，在第二范式的基础上，如果R的所有非主属性都直接函数依赖于它的候选键，则R是第三范式（3NF）。

例， SL(Sno, Sdept, Sloc) ∈ 2NF　SL(Sno, Sdept, Sloc) ∉ 3NF

SD（Sno， Sdept） ∈ 3NF　DL（Sdept， Sloc）∈ 3NF

- 若R∈3NF，则R的每一个非主属性既不部分函数依赖于候选码也不传递函数依赖于候选码。
- 如果R∈3NF，则R也是2NF。
- 采用模式分解法将一个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，并不能完全消除关系模式中的各种异常情况和数据冗余。

例：R（S，T，C），F={T → C，SC → T，ST →C}
判断最高属于第几范式？

- 候选码为：
- (S，T)和(S，C)
- S、T、C都是主属性，没有非主属性。所以R∈3NF
- 仍然存在插入异常等问题。

四、 BC范式（BCNF）

设关系模式R<U，F>∈1NF，如果对于R的每个非平凡函数依赖X→Y的左部（决定因素）X中必含有候选码，那么R是Boyce/Codd范式，简记为BCNF（改进的3NF）。

▪等价于：每一个决定属性因素都包含码

BCNF的含义

- ❖非主属性对候选键完全函数依赖
- ❖非主属性不传递依赖于任何一个候选键
- ❖主属性对不含它的候选键完全函数依赖
- ❖主属性不传递函数依赖于任何一个候选键

$$BCNF \subseteq 3NF \subseteq 2NF \subseteq 1NF$$

例1：R（S，T，C），F={T → C，SC → T，ST →C}是BCNF?

- 候选码为：
- (S，T)和(S，C)
- S、T、C都是主属性，没有非主属性。所以R∈3NF
- 函数依赖T → C的决定因素不是候选键，所以R不是BCNF

例2：关系模式SCG(S,C,G)中，S是学生，C表示课程，G表示名次。

每一个学生选修每门课程的成绩有一定的名次，每门课程中每一名次只有一个学生（即没有并列名次）。由语义可得到下面的函数依赖:

(S，C)→G ；(C，G)→S

候选码为:(S，C)与(C，G)

- 这两个码各由两个属性组成，而且它们是相交的。这个关系模式中显然没有属性对码传递依赖或部分依赖。所以SCG∈3NF，而且除(S,C)与(C,G)以外没有其它决定因素，所以SCG∈BCNF

3NF与BCNF的关系与区别

- 如果关系模式R∈BCNF，必定有R∈3NF
- 如果R∈3NF，且R只有一个候选码，则R必属于BCNF。

- 一个模式中的关系模式如果都属于BCNF，那么在函数依赖范畴内，它已实现了彻底的分离，已消除了插入和删除的异常。
- 3NF的“不彻底”性表现在可能存在主属性对码的部分依赖和传递依赖。

规范化总结

- 规范化的目的：解决插入、删除、更新异常以及数据冗余度高的问题；
- 规范化的方法：通过模式分解，从模式中个属性间的函数依赖着手，尽量做到每个模式表示客观世界中的一个“事件”；

❖ 关系模式规范化的基本步骤

消除非码的决定因素

1NF

↓ 消除非主属性对码的部分函数依赖

2NF

↓ 消除非主属性对码的传递函数依赖

3NF

↓ 消除主属性对码的部分和传递函数依赖

BCNF

综合例子

例1：试问下列关系模式最高属第几范式，并解释其原因

1）R{(A,B,C,D) ,(B→ D,(A,B)→ C)}

方法：

第一步：确定候选码

(A,B)

第二步：判断是否满足BCNF(即判断决定因素是否含有码)；

第三步：判断非主属性是否满足完全函数依赖和直接函数依赖。

非主属性（C,D）, B→D,(A,B) →D)

所以：R ∈1NF

试问下列关系模式最高属第几范式，并解释其原因

1）R{(A,B,C,D) ,(A→ C, (C,D)→ B)}

2）R{(A,B,C,D) ,(A→ C, D → B)}

3）R{(A,B,C) ,(A → B, B→ A ,A → C}

1）第一步：确定候选码 (A,D)

第二步：判断是否满足BCNF(即判断决定因素是否含有码)；

第三步：判断非主属性是否满足完全函数依赖和直接函数依赖。

非主属性（B,C），A→C,(A,D)→C)

所以：R ∈1NF

2）第一步：确定候选码 (A,D)

第二步：判断是否满足BCNF(即判断决定因素是否含有码)；

第三步：判断非主属性是否满足完全函数依赖和直接函数依赖。

非主属性（B,C），A→C,(A,D)→C)

所以：R ∈1NF

3）第一步：确定候选码 A，B

第二步：判断是否满足BCNF(即判断决定因素是否含有码)；

所以：R ∈BCNF

▪小结

了解数据不规范化带来的问题：
数据冗余、插入异常、删除异常、修改异常；

重点掌握几个概念：

- 函数依赖、非平凡和平凡函数依赖、部分和完全函数依赖、直接和传递函数依赖；
- 关系的候选码含义；
- 各种关系范式的含义：第一范式、第二范式、第三范式、BCNF

熟练掌握：给定某个关系模式能判别属于第几范式。

练习

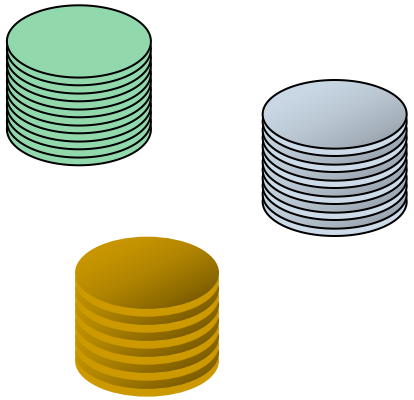
试问下列关系模式最高属第几范式，并解释其原因

1）R{(A,B,C,D) ,(AB → C,　　B → D)}

2）R{(A,B,C) ,(　A → B,　B → C)}

3）R{(A,B,C) ,(AB → C,　C → A}

4）R{(A,B,C) ,(AB → C,　BC → A}



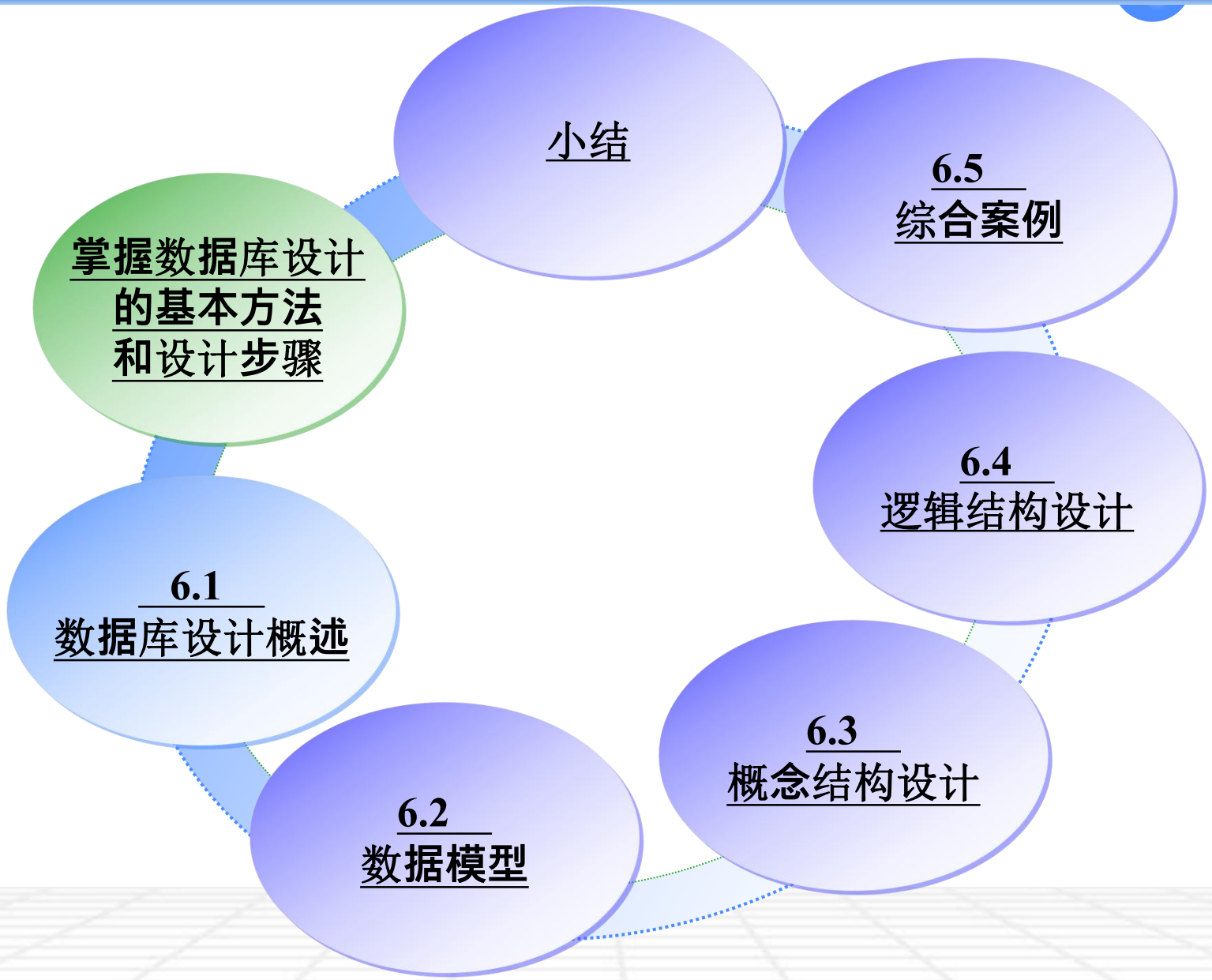
数据库系统

第六章 数据库设计

北京工业大学耿丹学院

计算机科学与技术专业

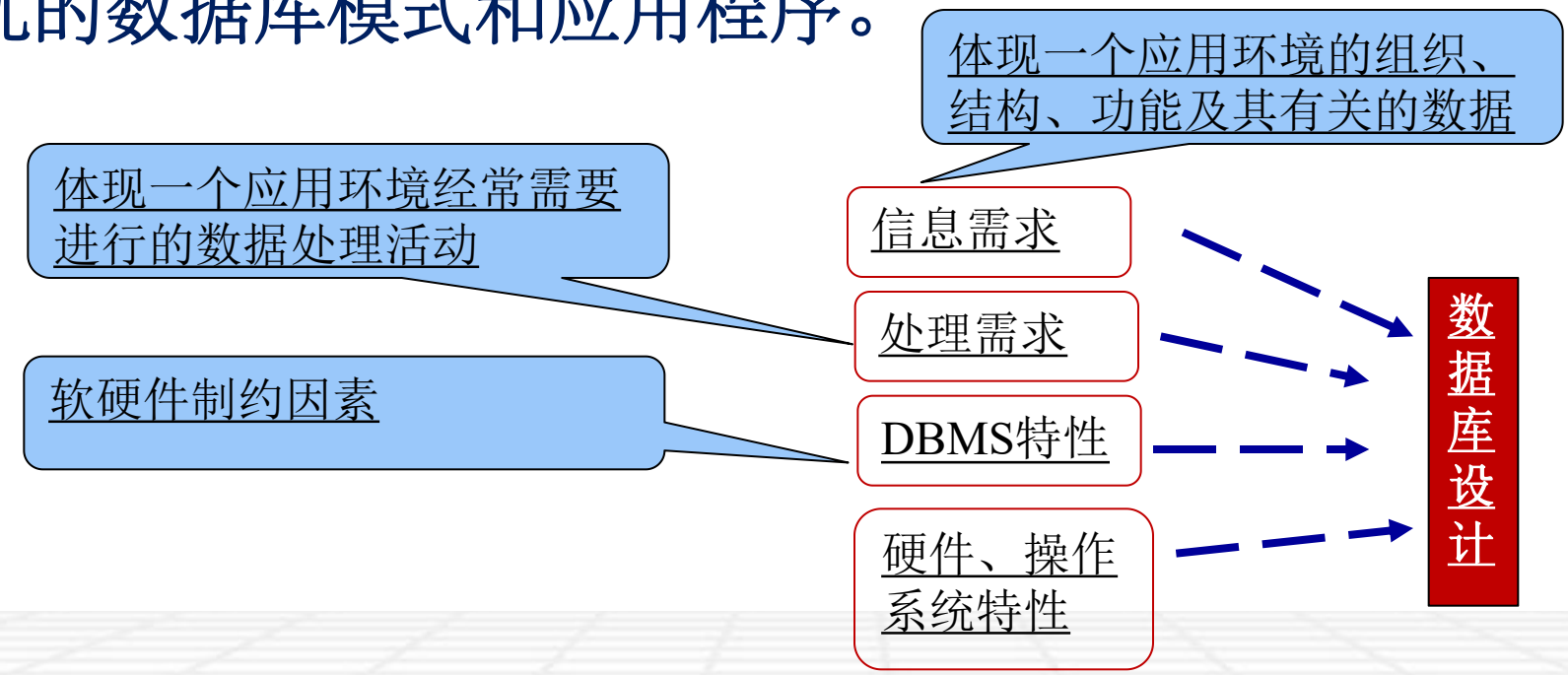
本章概述


小结
6.5
综合案例
掌握数据库设计
的基本方法
和设计步骤
6.4
逻辑结构设计
6.1
数据库设计概述
6.3
概念结构设计
6.2
数据模型

6.1 数据库设计概述

数据库设计的基本任务

根据一个应用环境的信息需求和处理需求，以及建立数据库所需的DBMS、操作系统和硬件的特性，设计出最优的数据库模式和应用程序。



6.1 数据库设计概述

数据库设计的特点

- 反复性：反复设计，逐步求精
- 多解性：设计结果不唯一，多重方案并存
- 分步进行：数据库设计分为多个阶段
- 结构设计和行为设计相结合：
  - 面向数据的设计方法（以信息需求为主）
  - 面向过程的设计方法（以处理需求为主）

6.1 数据库设计概述

数据库设计方法

- 新奥尔良法：运用软件工程的思想和方法进行数据库设计

  需求分析 概念设计 逻辑设计 物理设计

- 基于E-R模型的设计法：由Peter Chen在1976年提出的数据库设计方法，先用E-R图构造企业模式，然后再转换为特定数据库模式
- 基于3NF的设计法：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，然后再分解成若干个3NF关系模式的集合。
- 基于视图的设计法：先为每个应用建立视图，再将这些视图合并为全局概念模式。

数据库设计步骤

- 规划阶段
  - 建立数据库的必要性及可行性分析
- 需求分析阶段
  - 分析客户的业务需求和数据处理需求
- 概念设计阶段
  - 建立独立于计算机和DBMS的概念模型：E-R图
- 逻辑设计阶段
  - 将E-R图转化为关系模式，并用规范化理论审核
- 物理设计阶段
  - 为逻辑模型建立一个完整的能实现的数据库结构，包括存储存储结构和存取方法
- 实现阶段
  - 建立数据库及编写应用程序，输入数据，调试程序并运行
- 运行维护阶段
  - 进行安全性、完整性的维护、性能监督与改进，
  - 功能的扩充、修改和升级。

6.2 数据模型

- 现有的DBS均是基于某种数据模型建立的，因此，了解数据模型的基本概念是学习数据库的基础。
- 数据模型是一种模型，是对现实世界数据特征的抽象。它是用来描述数据、组织数据和对数据进行操作的。
- 通俗地讲数据模型就是现实世界的模拟。

6.2 数据模型

- 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

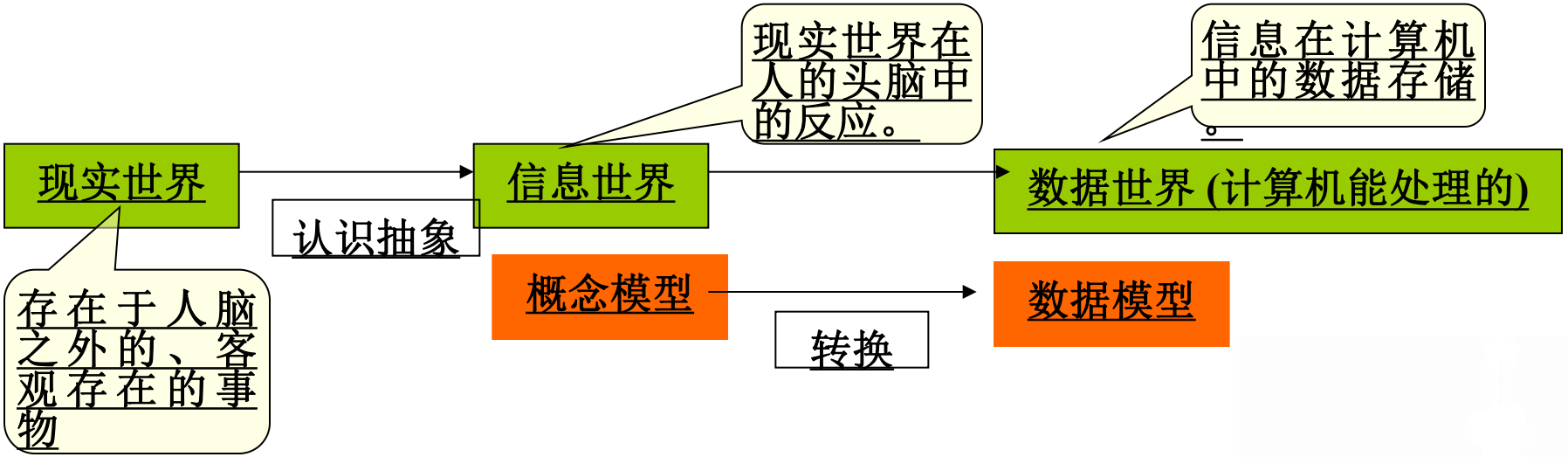
- 数据模型分成两个不同的应用层次

(1) 概念数据模型：也称信息模型，它是按用户的观点来对数据和信息建模，主要用于数据库设计，是独立于计算机系统的数据模型。

(2) 逻辑数据模型：也称为结构数据模型，简称为数据模型。是按计算机系统的观点对数据建模，主要用于DBMS的实现。分为：层次数据模型、网状模型、关系模型等。

6.2 数据模型

- 客观对象的抽象过程---两步抽象
  - 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。



6.3 概念结构设计

1. 概念模型概述

2. E-R模型

3. 基于E-R模型的概念结构设计

6.3.1 概念模型概述

- 概念模型的用途
  - 概念模型用于信息世界的建模
  - 现实世界到机器世界的一个中间层次
  - 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知识
  - 简单、清晰、易于用户理解。
- 常用的概念模型：E-R模型
- E-R模型的特点：现实世界是由实体和实体间的联系构成的。

6.3.2 E-R模型

E-R（Entity-Relationship）模型又叫**实体-联系模型**

- **1976年Peter Chen提出**
- **基本观点：现实世界是由实体和实体间的联系构成的。**

主要涉及两个方面：

(1) 如何将现实世界抽象为E-R模型中的各种元素

(2) 如何将它们用E-R图的形式正确的表达出来

6.3.2 E-R模型

E-R模型中涉及的基本概念（续）

(1) 实体（Entity）

客观存在并可相互区别的事物。
可以是具体的人、事、物或抽象的概念。

例如： 一名学生　一个部门　一名职工
　　部门的一次订货　学生的一次选课…

(2) 属性（ Attribute ）

实体所具有的某一特性。

例如： 学号 、姓名、性别、出生年份、系、入学时间
（980121 ，张三，男，1980年10月，计算机，1998）

6.3.2 E-R模型

E-R模型中涉及的基本概念（续）

(3) 码（ Key ）

能够唯一标识实体的最小的属性集。

例如： 学号 ---学生实体的码
（980121 ----代表学生张三）

(4) 域（ Domain ）

属性的取值范围。

例如： 性别（男，女）
学号（6位整数）

6.3.2 E-R模型

E-R模型中涉及的基本概念（续）

(5) 实体型（ Entity Type ）

对同类实体的抽象与刻画。 用实体名和属性名来表示。

例如：学生（学号 、姓名、性别、出生年份、系、入学时间）
顾客（编号、 姓名、年龄、电话）

(6) 实体集（ Entity Set ）

同类型实体的集合。

例如： 所有课程实体组成了一个实体集。
实体型和实体集是型和值的关系。

6.3.2 E-R模型

E-R模型中涉及的基本概念(续)

(7) 联系（Relationship）

a)实体内部之间的联系：组成实体的各属性之间的联系

b)实体之间的联系：不同实体型之间的联系

：同一实体型内部各实体之间的联系

- 一对一联系（1：1）
- 一对多联系（1：n）
- 多对多联系（m:n）

6.3.2 E-R模型

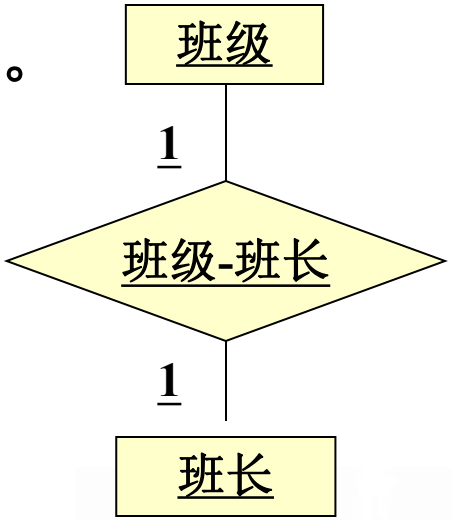
a). 不同实体集间的联系

- 一对一联系
  - 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。
  - 实例

    班级与班长之间的联系：

    一个班级只有一个正班长

    一个班长只在一个班中任职



1:1联系

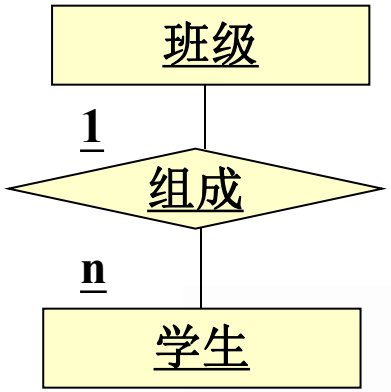
6.3.2 E-R模型

a). 不同实体集间的联系(续)

- 一对多联系
  - 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
  - 实例：班级与学生之间的联系：

    一个班级中有若干名学生，

    每个学生只在一个班级中学习



1:n联系

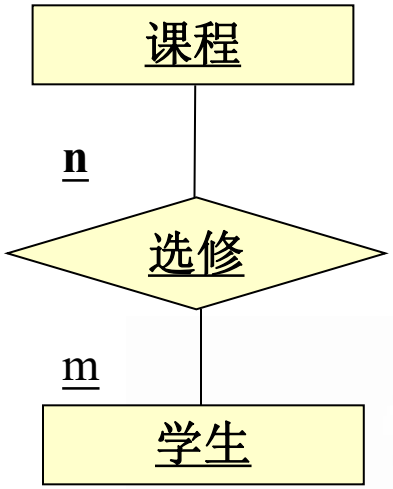
6.3.2 E-R模型

a) .不同实体集间的联系(续)

- 多对多联系（m:n）
  - 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
  - 实例：课程与学生之间的联系：

    一门课程同时有若干个学生选修

    一个学生可以同时选修多门课程



m:n联系

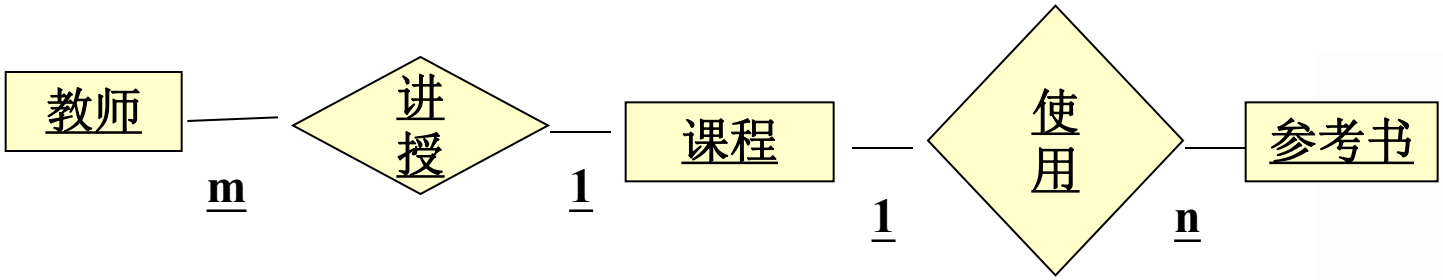
6.3.2 E-R模型

练习：

如果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。

找出实体及实体间的联系，并指出联系的类型

课程与教师、参考书之间的联系是一对多的。



6.3.2 E-R模型

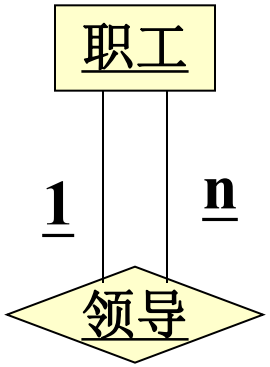
b). 同一实体集内各实体间的联系

❖ 一对多联系

- 实例

职工实体集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。



同一实体型内部的1:n联系

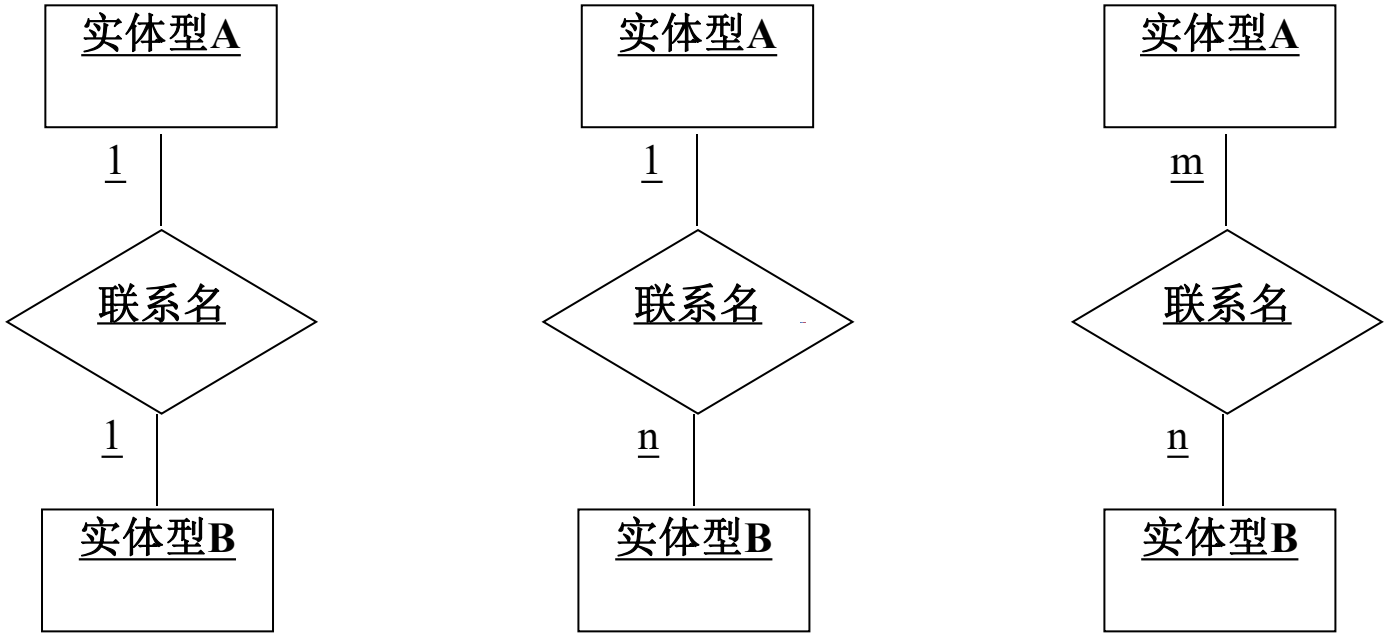
6.3.2 E-R模型

E-R模型的表示方法:E-R 图

- E-R模型以E-R图的形式表现，在E-R图中：
  - 实体：用矩形表示，矩形框内写明实体名。
  - 属性：用椭圆形(带半圆的矩形框)表示，椭圆形(带半圆的矩形框)内写明属性名。并用无向边与对应的实体连接。当某一个属性或属性组合指定为主码时，要在该属性或属性组合名下面画一下划线作为标志，（或者是在实体和属性的连线上标记一斜线）。
  - 联系：用菱形表示，菱形框内写明联系名，并用无向边与有关实体连接起来，同时在无向边旁边标上联系的类型。
  - 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。

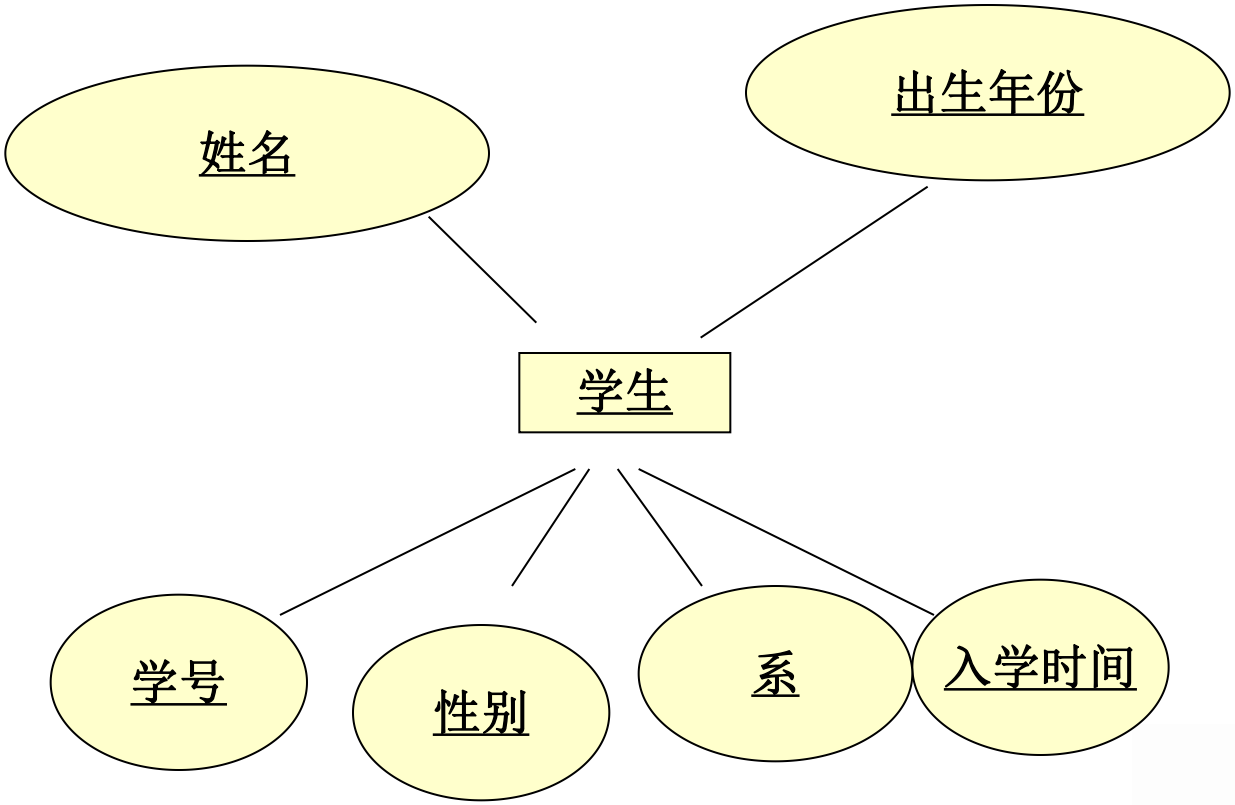
6.3.2 E-R模型

- 实体联系图



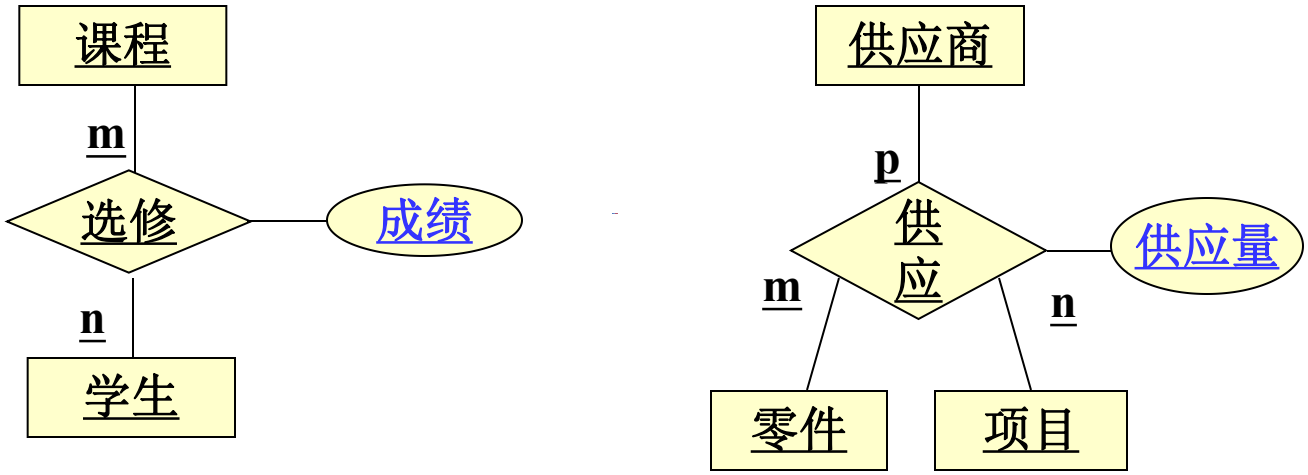
（a）1：1联系　　（b）1：n联系　　（c）m：n联系

6.3.2 E-R模型



学生实体及属性

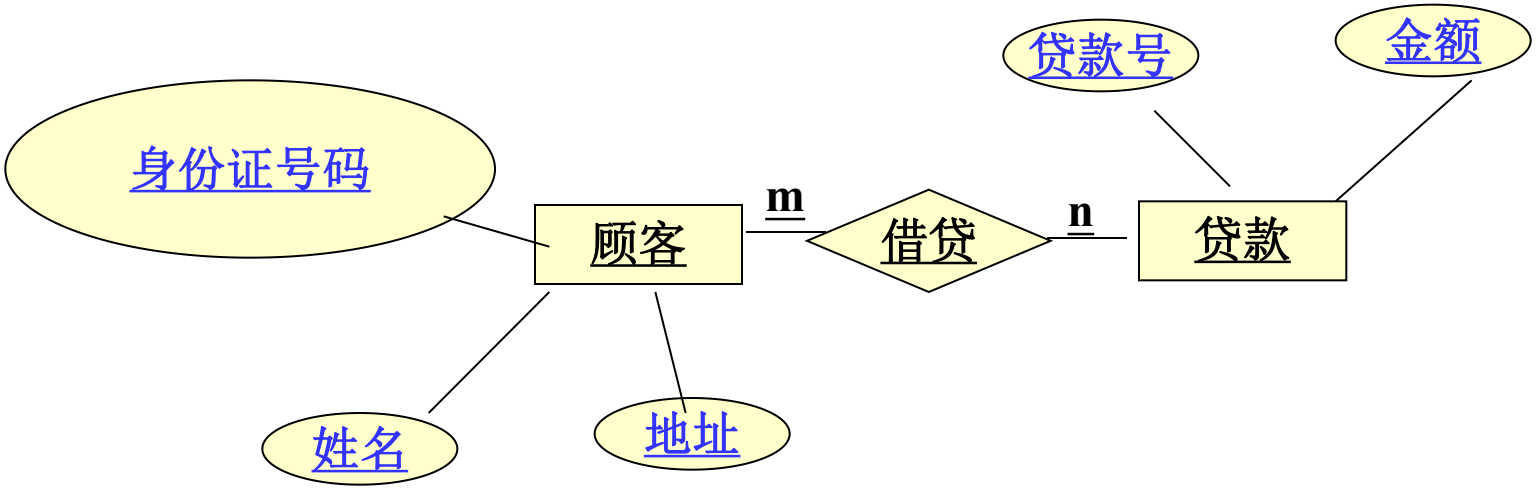
6.3.2 E-R模型



实体型间联系的属性

6.3.2 E-R模型

- 实例1：用E-R图来表示顾客借贷的概念模型。



顾客借贷联系图

E-R模型设计原则

如何正确的建模

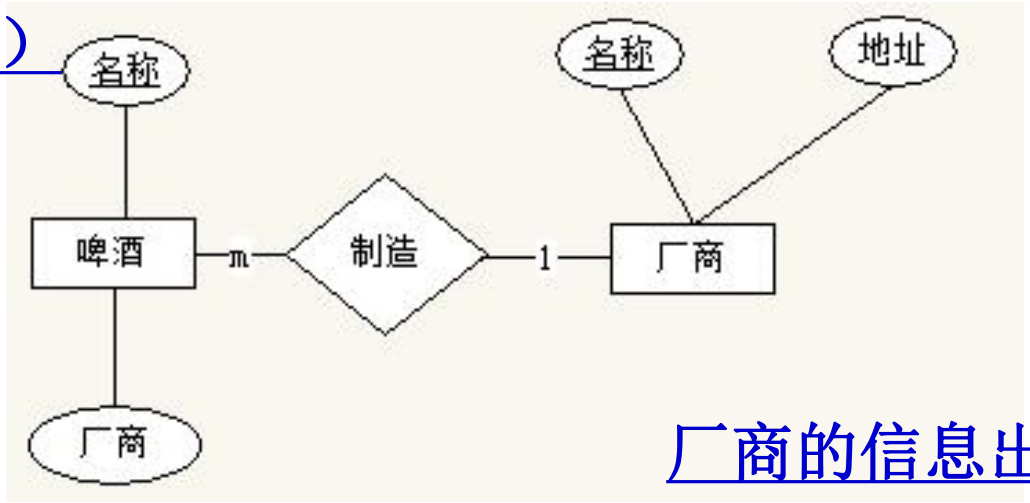
- 根据应用的语义和信息需求来决定哪些事物需要被抽象建模？
- 在模型中应该被抽象成实体、属性、联系？

E-R模型设计原则

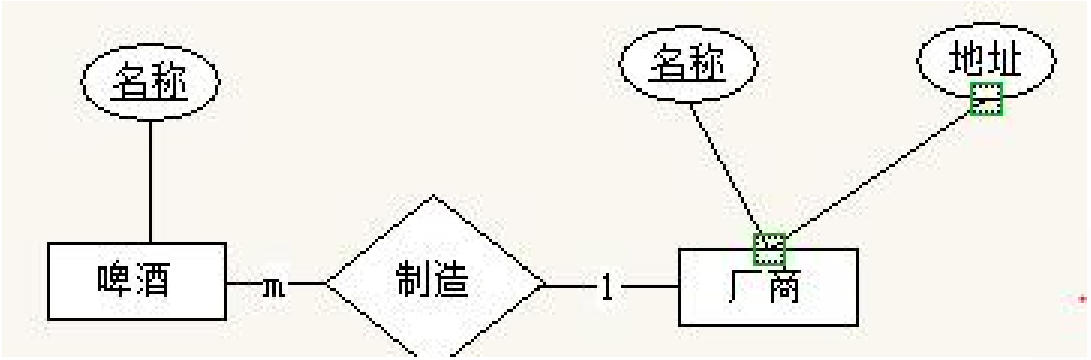
- E-R设计中应该避免冗余，避免浪费空间和操作异常。
- 一般情况下，凡能作为属性对待的，应尽量作为属性，以简化E-R图的处理。
- 如果作为实体对待，则应该满足以下条件中的一个：
  - ① 该实体具有除码之外的其他属性；
  - ② 该实体是某个一对多或多对多联系的“多”端。

E-R模型设计原则—实例

避免冗余（一）

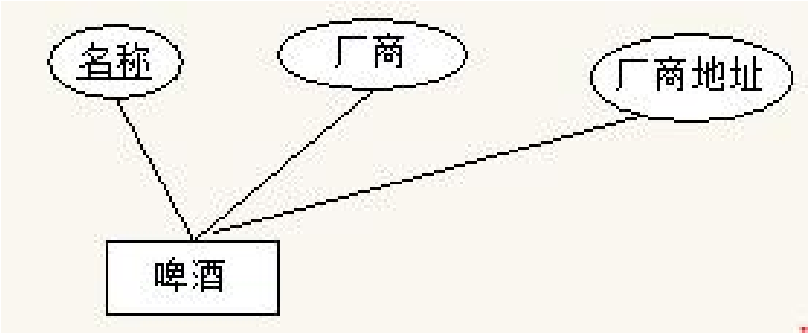


厂商的信息出现了两次！

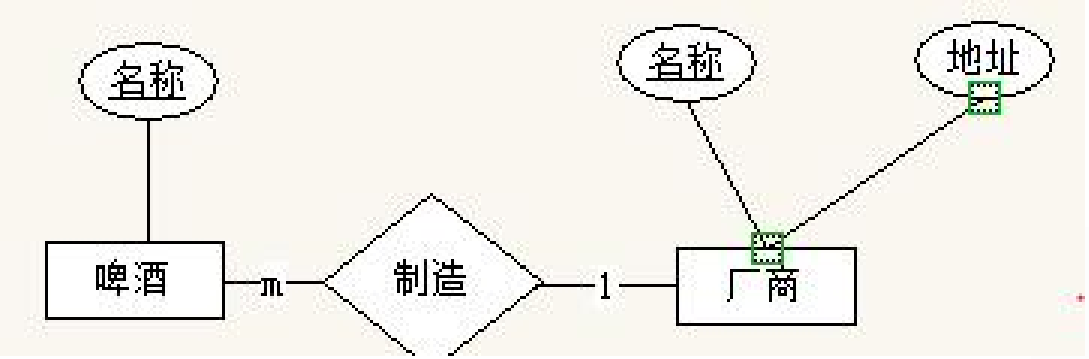


E-R模型设计原则—实例

避免冗余（二）

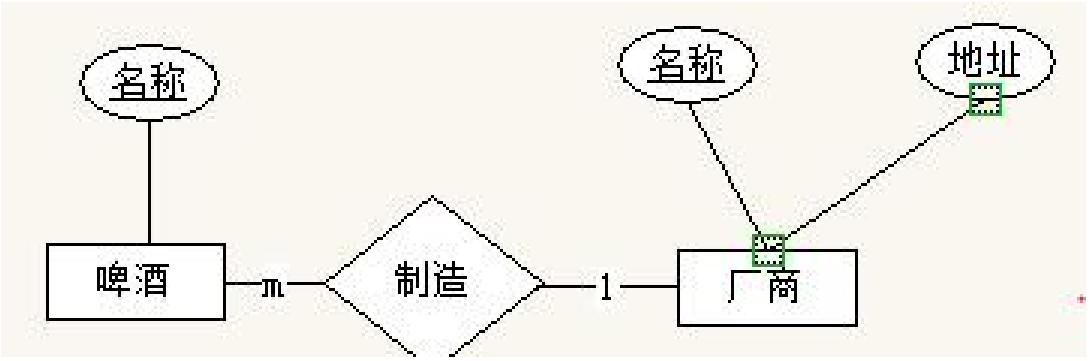


同一厂商的地址可能多次存储！

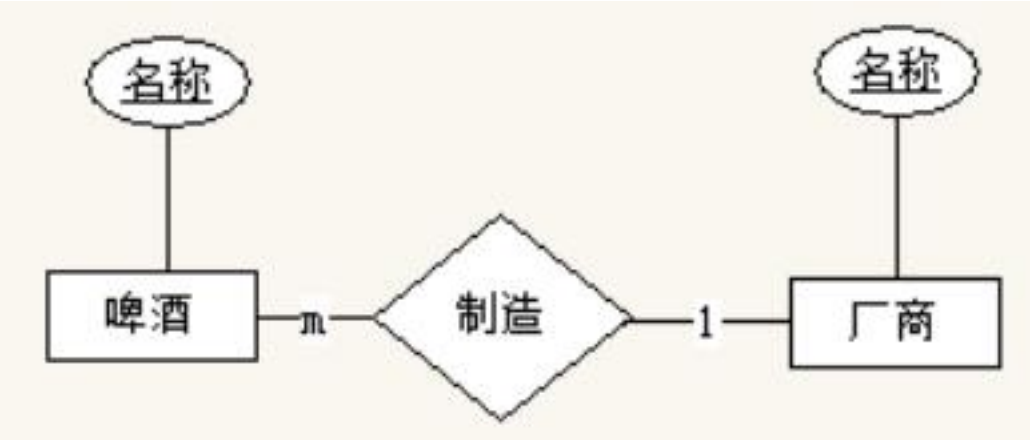


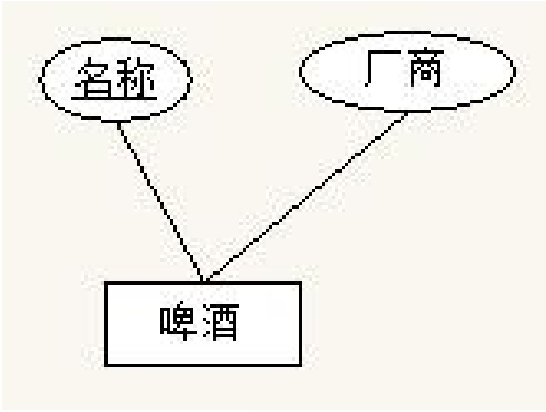
E-R模型设计原则—实例

实体或属性？



- 啤酒虽然只有一个属性，但是它是一对多联系的“多”端
- 厂商虽然是一对多联系的一端，但是它有多个属性。





- 厂商只有一个属性，并且它是一对多联系的“一”端。

6.3.3 概念结构设计

设计概念结构的E-R模型可采用四种方法：

（1）自顶向下：先确定全局E-R模型的框架，再逐步细化

（2）自底向上：先完成各局部应用的E-R模型，然后再它们集成全局E-R模型。

（3）逐步扩张：先确定最重要的核心E-R模型，然后向外扩充，以滚雪球的方式逐步生成完整的E-R模型

（4）混合策略：采用自顶向下和自底向上相结合的方式，先用自顶向下定义全局框架，再以它为骨架来集成自底向上方法中涉及的各个局部E-R模型。

最常用的方法是自底向上设计法，包括两个步骤：

（1）设计局部的E-R模型，即设计用户视图；

（2）集成各局部E-R模型，形成全局E-R模型

概念结构设计—实例

步骤一：设计局部E-R模型

例：在一个教务管理系统中，有如下语义约束：

① 一个学生可选修多门课程，一门课程可为多个学生选修；

② 一个教师可讲授多门课程，一门课程可有多个教师讲授；

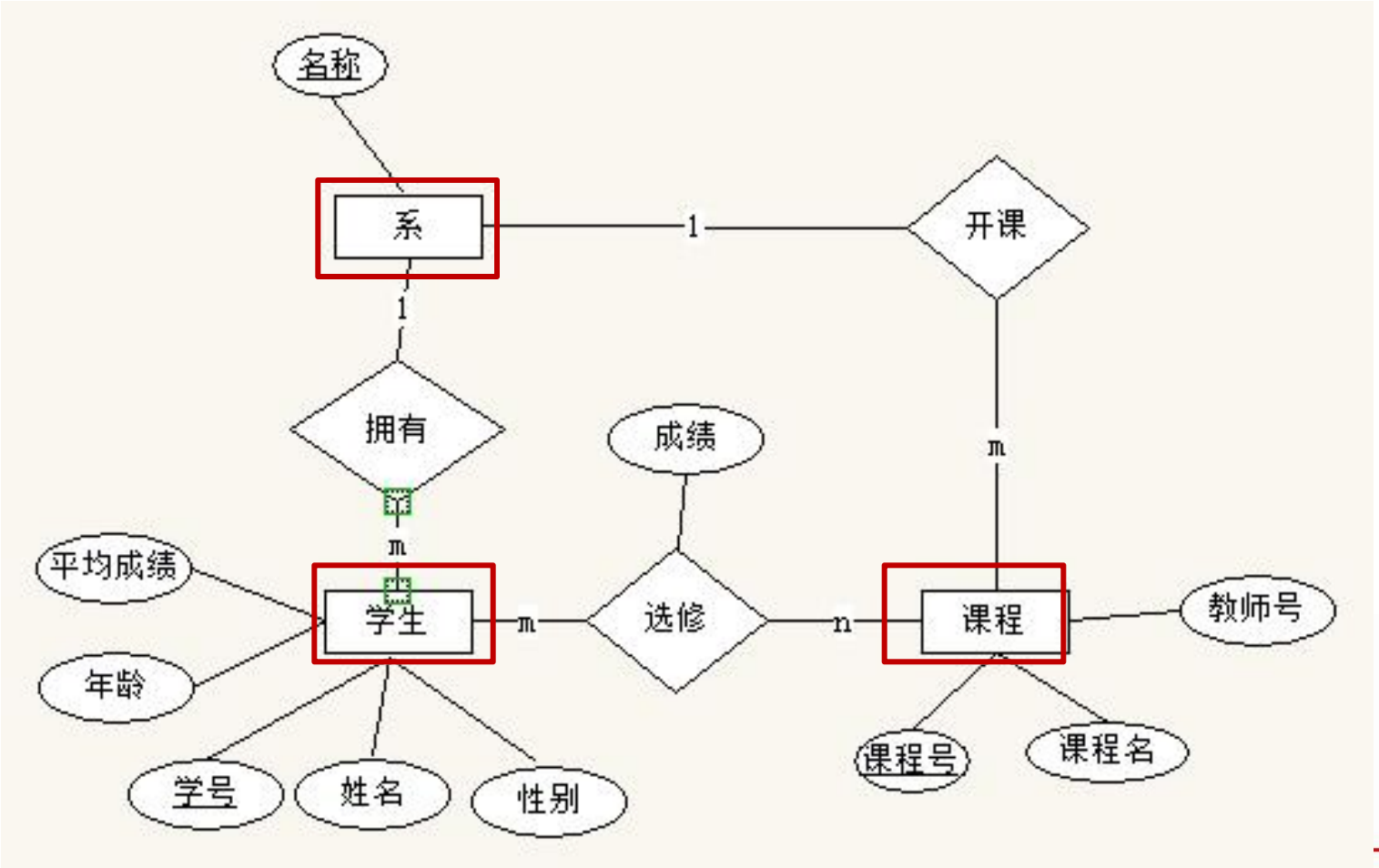
③ 一个系可由多个教师，一个教师只能属于一个系；

④ 一个系可有多个学生，一个学生只能属于一个系；

⑤ 学生可以选修非本系开设的课程，但是教师只讲授本系开设的课程。

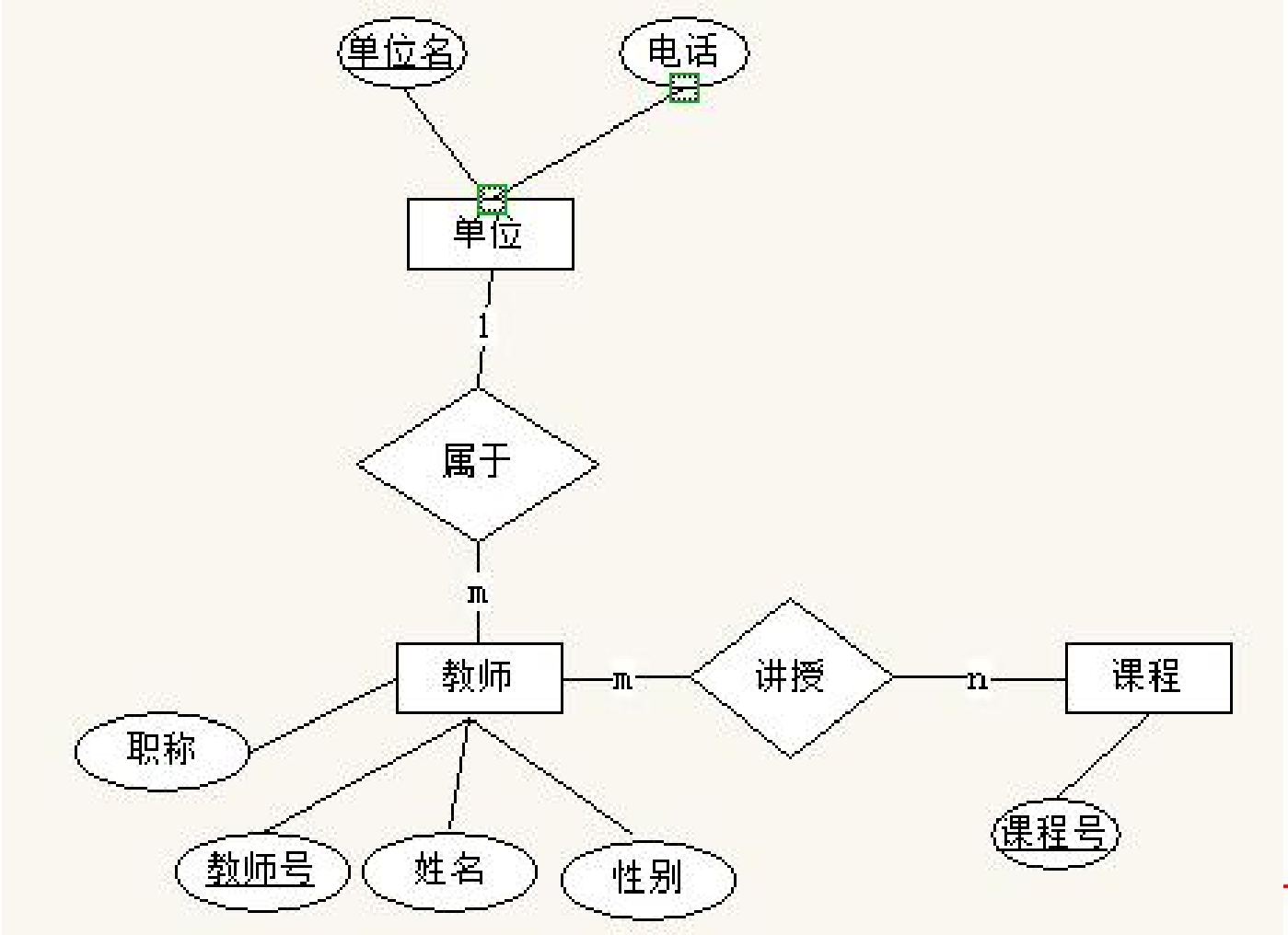
概念结构设计—实例

学生选课局部E-R图（学生、课程、系）



概念结构设计—实例

教师任课局部E-R图



概念结构设计—实例

各局部E-R图之间存在的不一致现象，称为冲突，合并局部E-R图的关键在于消除冲突

步骤二：全局E-R模型设计

（1）合并局部E-R图，消除冲突，生成初步E-R图

① 属性冲突

属性域冲突-----属性值的类型、取值范围或取值集合不同。

比如年龄，有的用生日表示，有的用岁数来表示。

属性值单位冲突-----比如服装尺码，有的以厘米为单位，有的以尺寸为单位

属性冲突属于用户业务上的约定，必须与用户协商后解决。

概念结构设计—实例

步骤二：全局E-R模型设计

（1）合并局部E-R图，消除冲突，生成初步E-R图

② 命名冲突

同名异义-----即不同意义的对象在不同局部应用中具有相同的名字。

例如：班级和系之间的联系，班级与学生之间的联系，都命名为“属于”；

异名同义-----即同一意义的对象在不同局部应用中具有不同的名字。

例如：科研项目在有的部门称为项目，在有的部门称为课题

命名冲突也是属于用户业务上的约定，必须与用户协商后解决。

概念结构设计—实例

③ 结构冲突

同一对象在不同应用中有不同的抽象。

例如，教师的职称在某一局部应用中被当成实体，而在另一局部应用中被当做属性；

解决方法：统一抽象级别

同一实体在不同应用中属性组成不同。

解决方法：取该实体在各应用中属性的并集，再适当调整属性的次序

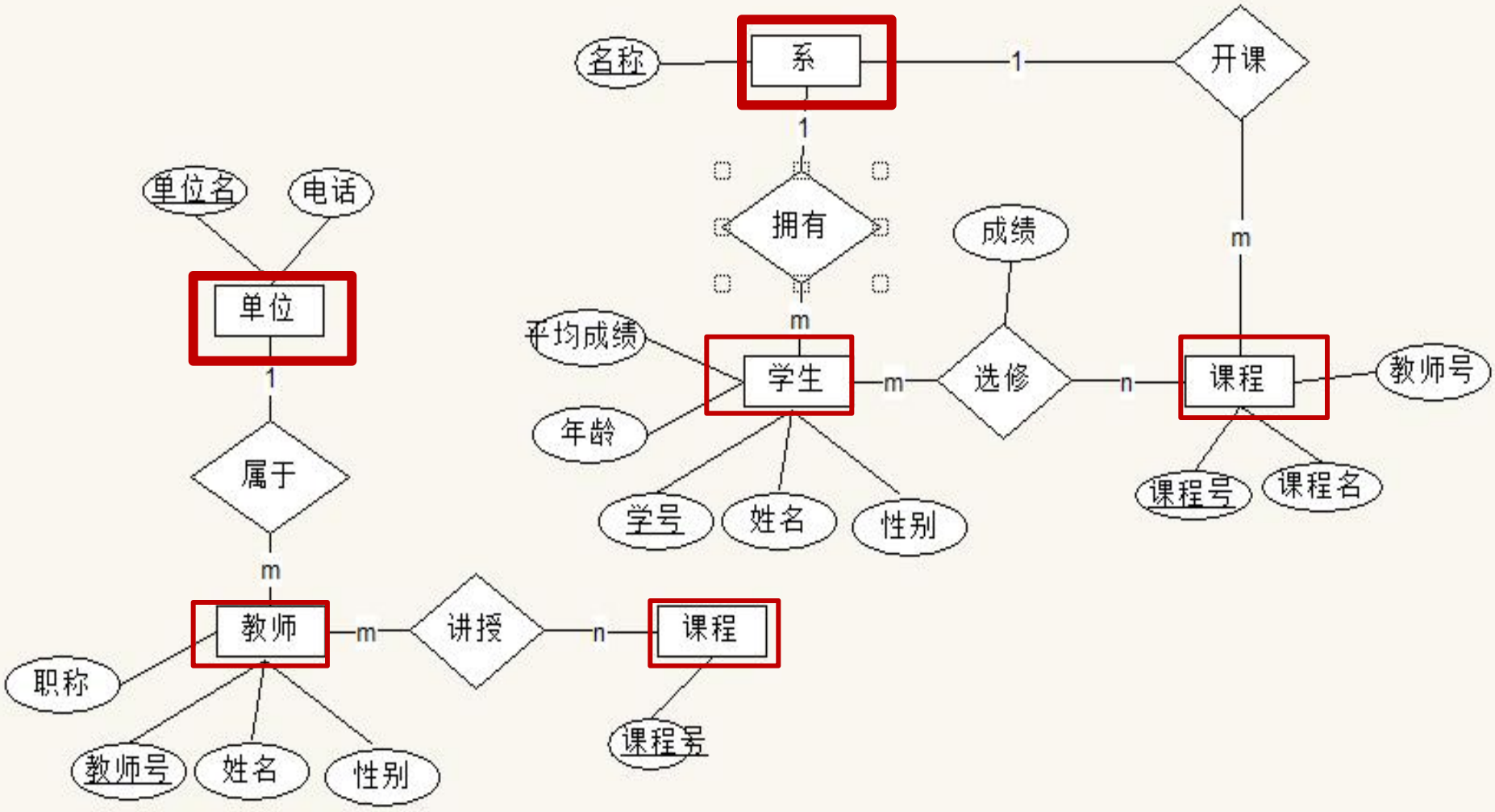
同一联系在不同应用中呈现不同的类型。

比如E1与E2在某个应用中可能是一对多联系，在另一应用中可能是多对多联系，或者是E1,E2,E3三者之间的三元联系

解决方法：根据应用语义对实体联系的类型进行综合调整。

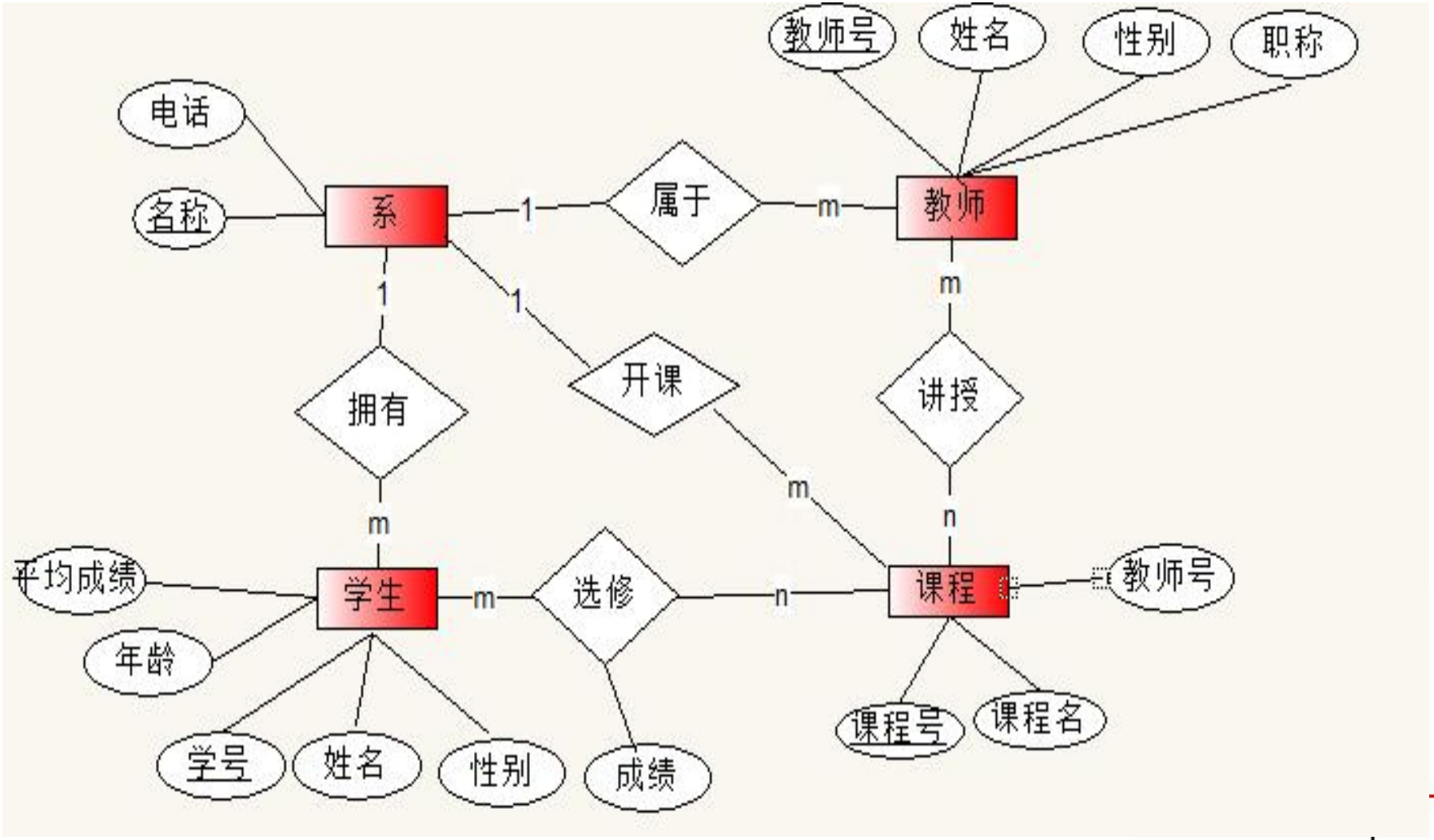
概念结构设计—实例

局部E-R图合并举例：



概念结构设计—实例

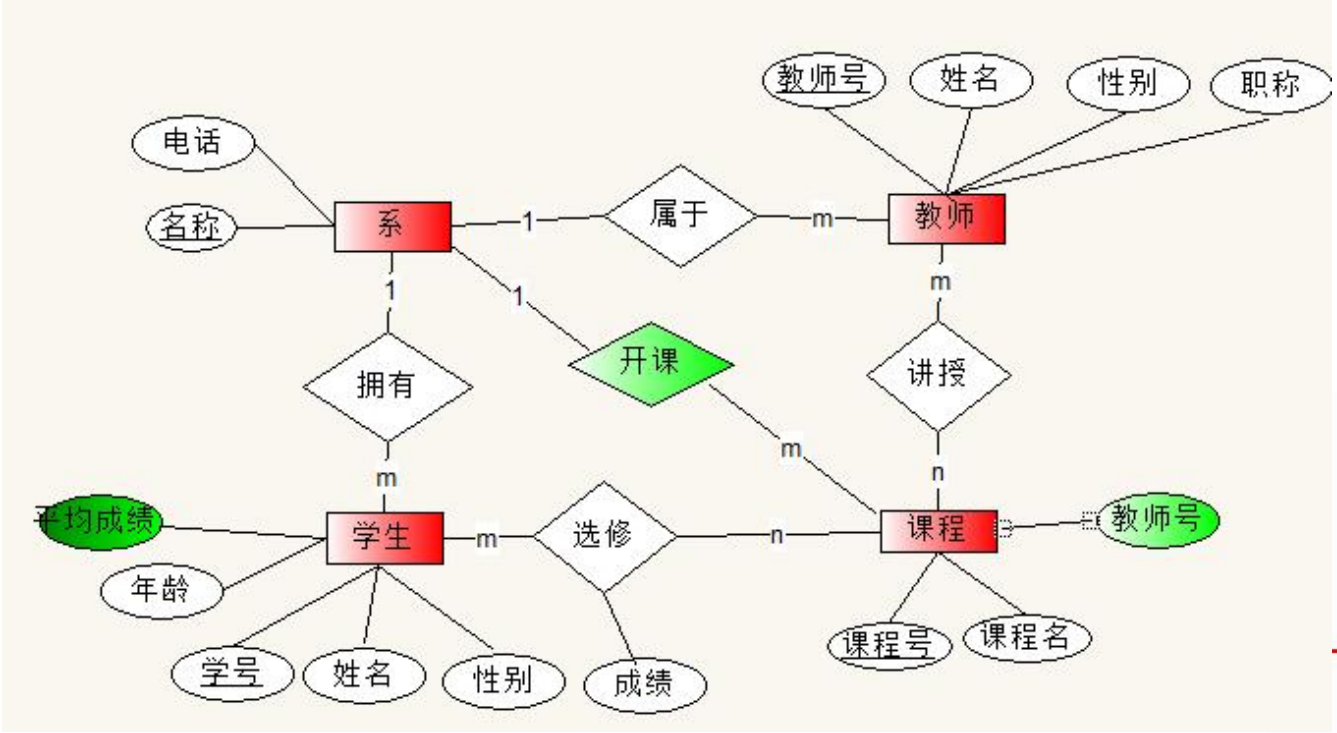
初步E-R图：



概念结构设计—实例

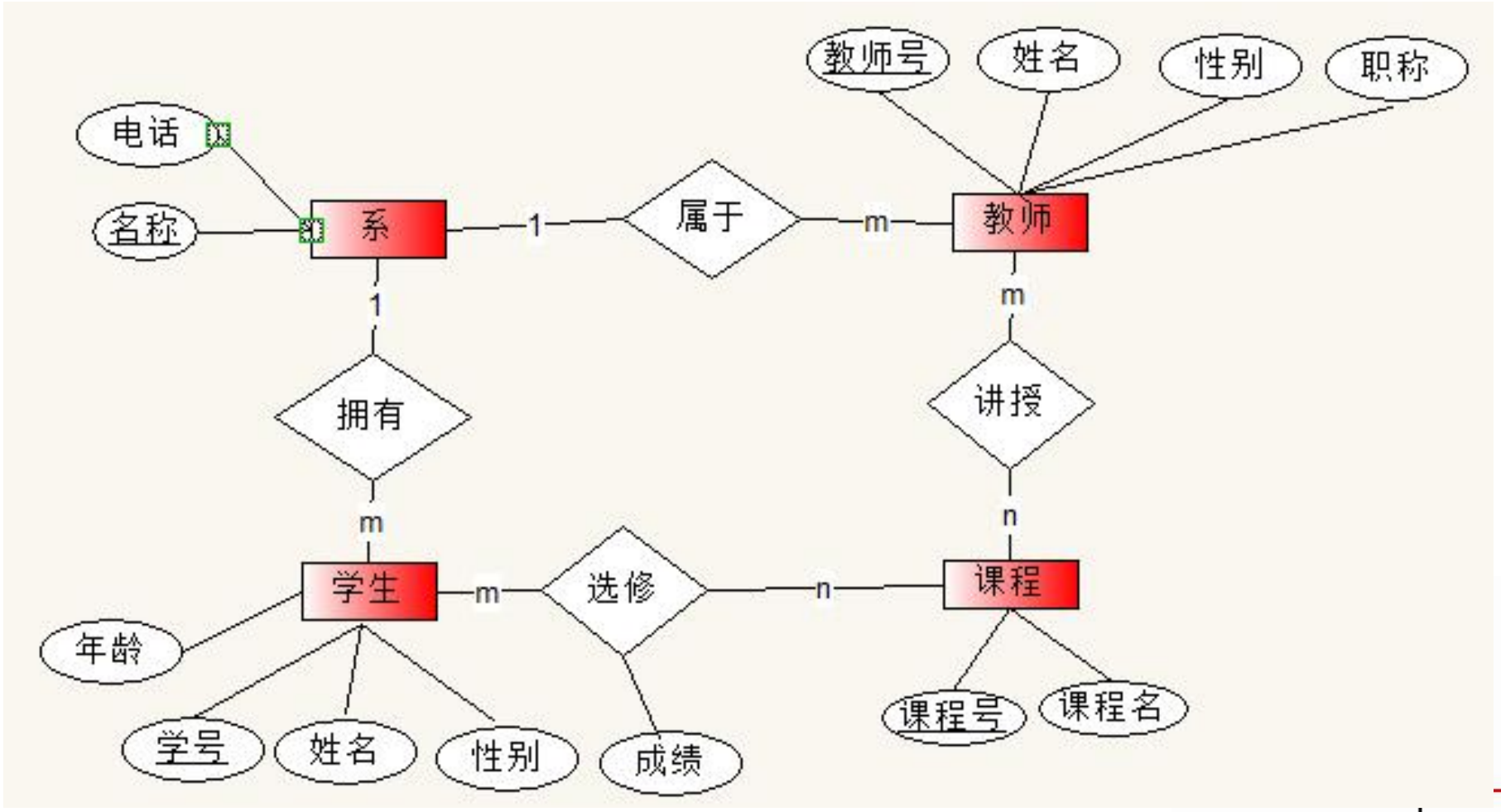
（2）消除不必要的冗余，生成基本E-R图：

- 冗余包括冗余的数据和冗余的联系：
  - 冗余的数据是指可由基本数据导出的数据；
  - 冗余的联系是指可由其他联系导出的联系



概念结构设计—实例

基本E-R图：



6.4 逻辑结构设计

逻辑结构设计的任务和步骤

概念结构设计 → E-R模型

面向用户的模型

它独立于具体的DBMS和计算机系统，便于设计人员与用户交流

为了在计算机系统中建立数据库

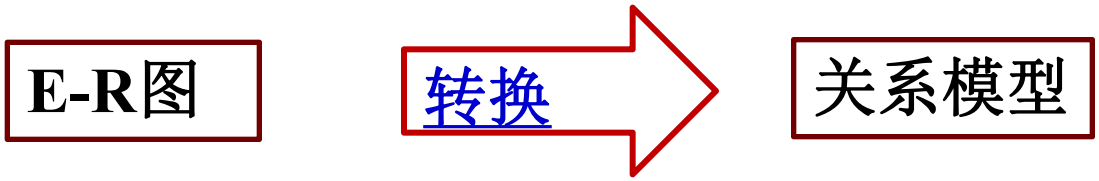
数据库逻辑设计的任务是将概念结构转换成特定的DBMS所支持的数据模型

逻辑结构设计的基本步骤

- 初始关系模式设计（E-R模型转换成关系模型）
- 关系模式优化（满足3NF或BCNF）
- 设计用户外模式（子模式）

6.4 逻辑结构设计

步骤一：初始关系模式设计（E-R模型转换成关系模型）



实体、属性、联系　　关系

实际上就是将实体、属性和联系转换为一种关系模式

6.4 逻辑结构设计

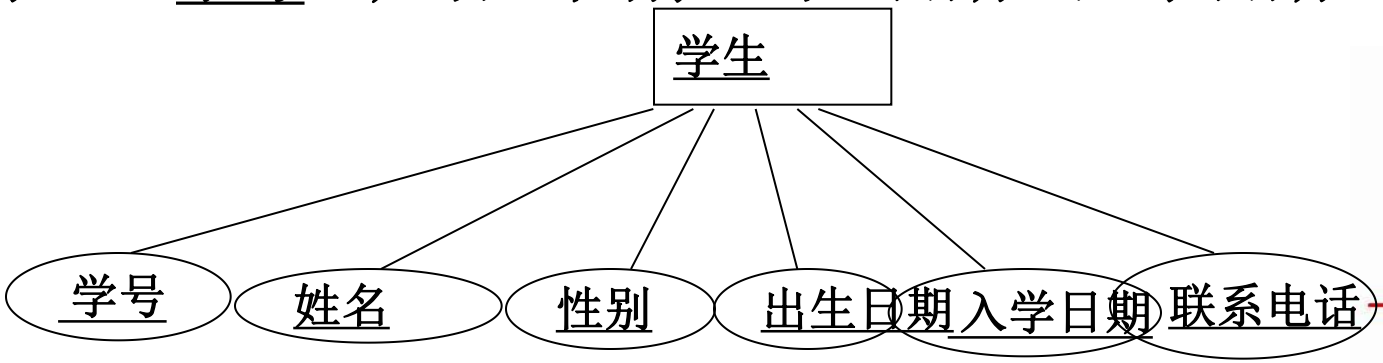
步骤一：初始关系模式设计

1. E-R图向关系模式转换的原则

- （1）实体转换关系模式。
  - 一个实体转换一个关系模式
  - 实体的属性转换为关系的属性
  - 实体的码转换为关系的码

例，学生实体可以转换为如下关系模式：

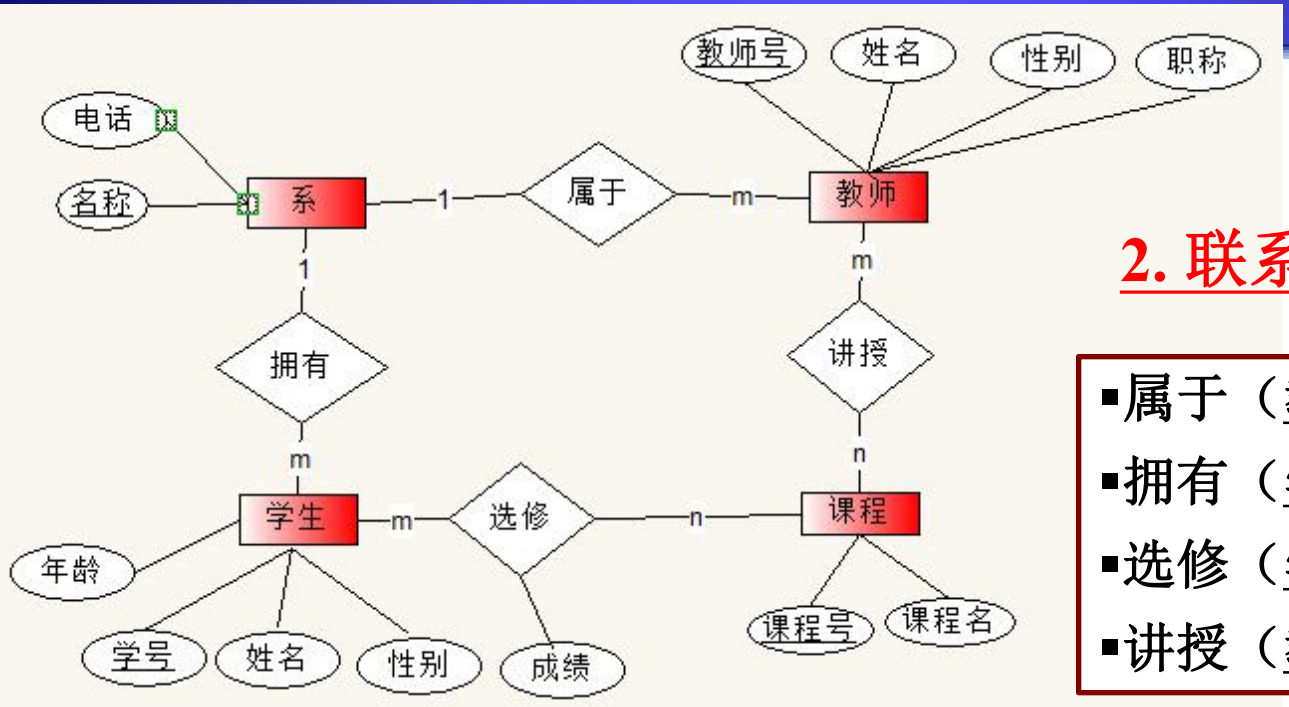
学生（学号，姓名，性别，出生日期，入学日期，联系电话）



6.4 逻辑结构设计

- （2）联系的转换。
  - 一个联系转换一个关系模式
  - 关系的属性
    - 与该联系相连的各实体的码
    - 该联系自身的属性
  - 关系的码：
    - 一对一的二元联系：两端实体的码都是候选码
    - 一对多的二元联系：“多”端实体的码
    - 多对多的二元联系：至少包含两端实体的码的并集
    - 存在“一”端的多元联系：除了“一”端以外的其他“多”端实体码的并集
    - 不存在“一”端的多元联系：至少包含所有相关实体码的并集
  - 关系的外码：
    - 因为关系的一部分属性来自相关联的实体的码，所以这些属性就是关系的外码

E-R图转换关系模式实例



1. 实体转换

- 学生（学号，姓名，性别，年龄）
- 教师（教师号，姓名，性别，职称）
- 系（系名，电话）
- 课程（课程号，课程名）

2. 联系转换

- 属于（教师号，系名）
- 拥有（学号，系名）
- 选修（学号，课程号，成绩）
- 讲授（教师号，课程号）

3. 确定联系关系中的外码约束

例如：属于（教师号，系名）

教师号，系名均为外码

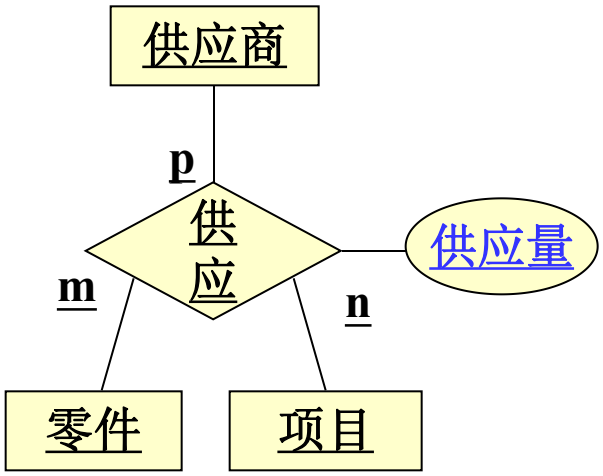
教师号参考教师关系中的教师号

系名参考系关系中的系名

……

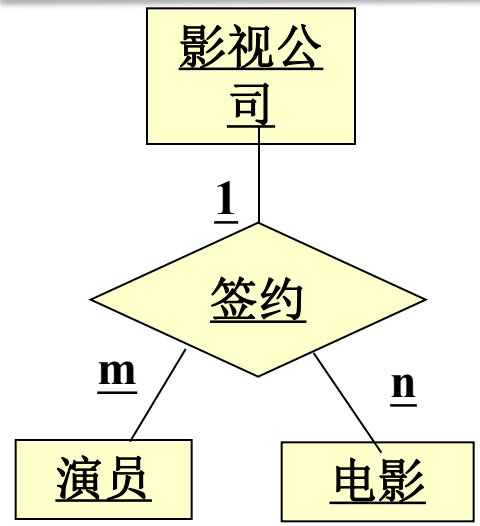
E-R图转换关系模式实例

多元联系转换



▪转换后供应关系的属性为三端实体的码+自身的属性

供应（供应商号、零件号、项目号，供应量）



转换后签约关系的属性为三端实体的码+自身的属性

但是关系的码为：多端实体的码

签约（演员编号，电影编号，公司编号）

E-R图转换关系模式实例

4.码相同的关系可以合并成一个关系

合并前

- 学生（学号，姓名，性别，年龄）
- 教师（教师号，姓名，性别，职称）
- 系（系名，电话）
- 课程（课程号，课程名）
- ~~属于（教师号，系名）~~
- ~~拥有（学号，系名）~~
- 选修（学号，课程号，成绩）
- 讲授（教师号，课程号）

合并后

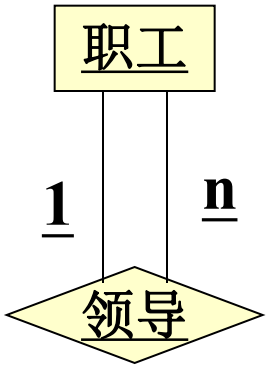
学生（学号，姓名，性别，年龄，系名）

教师（教师号，姓名，性别，职称，系名）

- 系（系名，电话）
- 课程（课程号，课程名）
- 选修（学号，课程号，成绩）
- 讲授（教师号，课程号）

E-R图转换关系模式实例

4.自身联系转换



同一实体型内部的
1:n联系

实体转换为一个关系：

职工（职工号，职工名）

联系转换为一个关系：

领导（职工号，领导者名）

合并联系关系

职工（职工号，职工名，领导者名）

6.4 逻辑结构设计

步骤一：初始关系模式设计（E-R图转换关系模式小结）

- （1）每个实体转换关系模式。
  - 关系的属性：实体的属性
  - 关系的码：实体的码
- （2）多对多联系转换关系模式。
  - 关系的属性：各端码+自身的属性
  - 关系的码：各端的码
- （3）一对多联系转换关系模式（一般合并对多端）。
  - 关系的属性：原关系的属性+联系自身的属性+1端的码
  - 关系的码：不变
- （4）一对一联系转换关系模式（一般合并对任意一端）。
  - 关系的属性：原关系的属性+联系自身的属性+另一1端的码
  - 关系的码：不变

6.4 逻辑结构设计

步骤二：关系模式优化

关系规范化理论

规范化设计

3NF或BCNF

应用规范化理论对关系模式进行规范化，以减少乃至消除关系模式中存在的各种异常，改善完整性、一致性和存储效率

反规范化设计

必要时进行反规范化设计，以提高查询效率。

根据应用的特点，对关系模式进行垂直分解和水平分解，以提高操作效率和存储空间利用率。

6.4 逻辑结构设计

步骤二：关系模式优化

水平分解

是把关系中的元组分解成若干个子集。

对于经常进行大量数据的分类条件查询的关系，可进行水平分解，这样可以减少每次查询需要访问的元组数。

例如：
关系模式 客户（账号，姓名，地区...）,如果绝大多数查询一次只涉及一个地区，就可以把整个关系按地区进行水平分解。

北京客户（账号，姓名，地区...）
山西客户（账号，姓名，地区...）
山东客户（账号，姓名，地区...）

6.4 逻辑结构设计

步骤二：关系模式优化

垂直分解

是把关系中的属性分解成若干个子集。

对于经常一起使用的属性放到一个子关系中。

例如：教师关系模式

教师（教师号，姓名，性别，年龄，职称，工资，住址，电话，简历）

如果经常查询前六项，后三项很少使用，则可以对教师关系进行垂直分解。

教师关系1（教师号，姓名，性别，年龄，职称，工资）

教师关系2（教师号，住址，电话，简历）

6.4 逻辑结构设计

步骤三：设计外模式

设计三级模式中的外模式

SQL中的视图

指导思想：注重局部用户的要求

使用更符合用户习惯的别名、计量单位等
从安全性出发，对不同级别的用户定义不同的视图
从易用性出发，将复杂查询语句定义为视图

6.5 数据库设计综合案例

数据库设计任务：

根据某一用户领域需求进行概念结构设计和逻辑结构设计以及规范化设计，要求：

（1）画出系统的E-R图。在E-R图中需注明实体的属性、联系的类型及联系的标识符。

（2）将E-R图转换为关系模式，并指出每个关系模式的主键和外键。

（3）在函数依赖范畴内分析每个关系模式达到第几范式，并说明理由，达不到3NF的要将其分解成满足3NF的关系模式。

6.5 数据库设计综合案例

用户需求描述：

某中学要对学校运动会进行计算机管理，信息系统拟对班级、运动员、比赛项目、裁判员等信息进行管理，管理规则如下：

（1）有若干班级，每个班级有若干运动员，运动员只能属于一个班。运动员的信息包括运动员号、姓名、性别、年龄。每个班级包括班级号、班级名、人数。

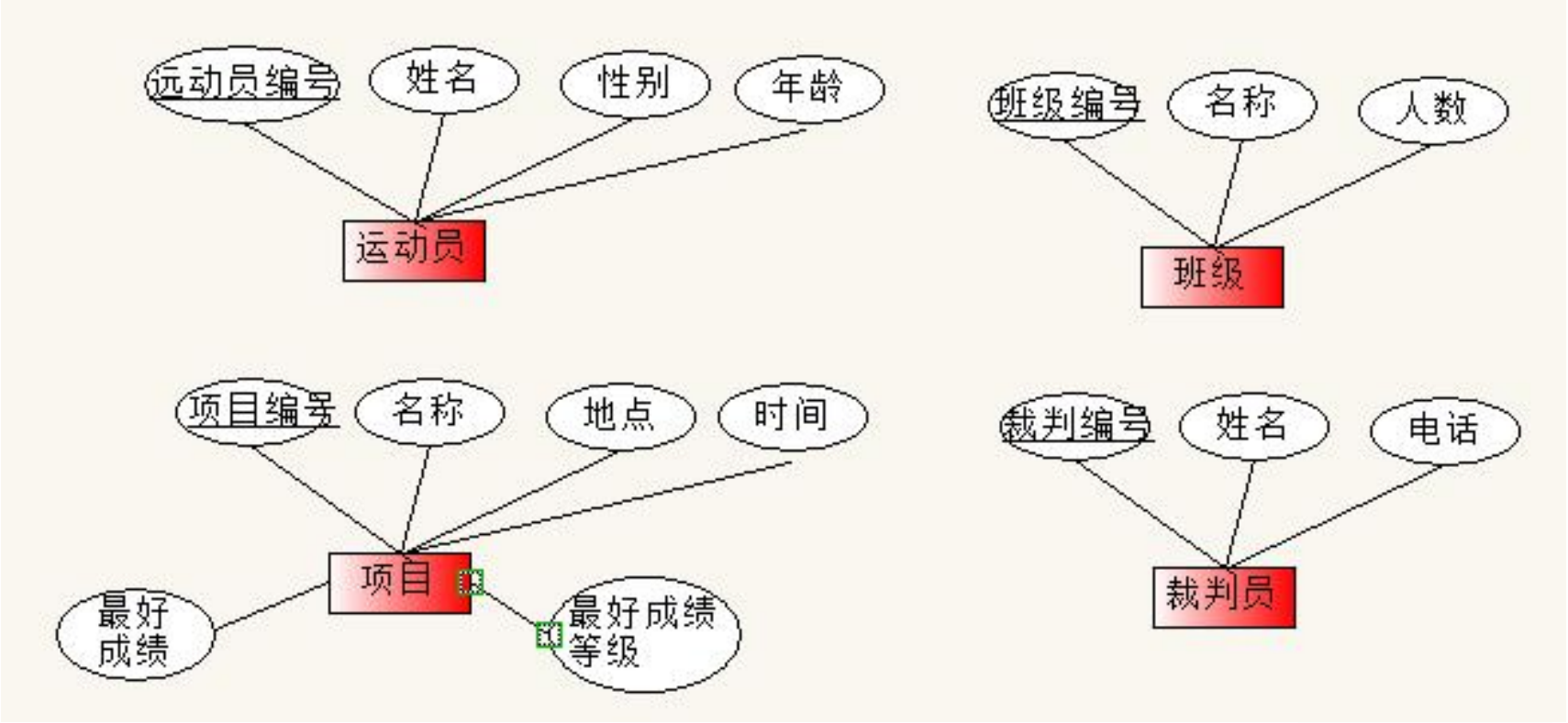
（2）每名运动员最多可参加三项比赛，系统需记录项目号、项目名、比赛地点、比赛时间、该运动项目的最好成绩及最好成绩等级。

（3）每名裁判员可做多项比赛的裁判，一项比赛需要两个裁判员。系统需记录裁判员号、裁判员姓名、电话等信息。

此外，系统为了计算运动员个人名次和团队总分，还要记录每个比赛项目的运动员成绩。

6.5 数据库设计综合案例

第一步：画出实体-属性图



6.5 数据库设计综合案例

第二步：分析实体之间的联系

一对多

（1）有若干班级，每个班级有若干运动员，运动员只能属于一个班。运动员的信息包括运动员号、姓名、性别、年龄。每个班级包括班级号、班级名、人数。

多对多

（2）每名运动员最多可参加三项比赛，系统需记录项目号、项目名、比赛地点、比赛时间、该运动项目的最好成绩及最好成绩等级。

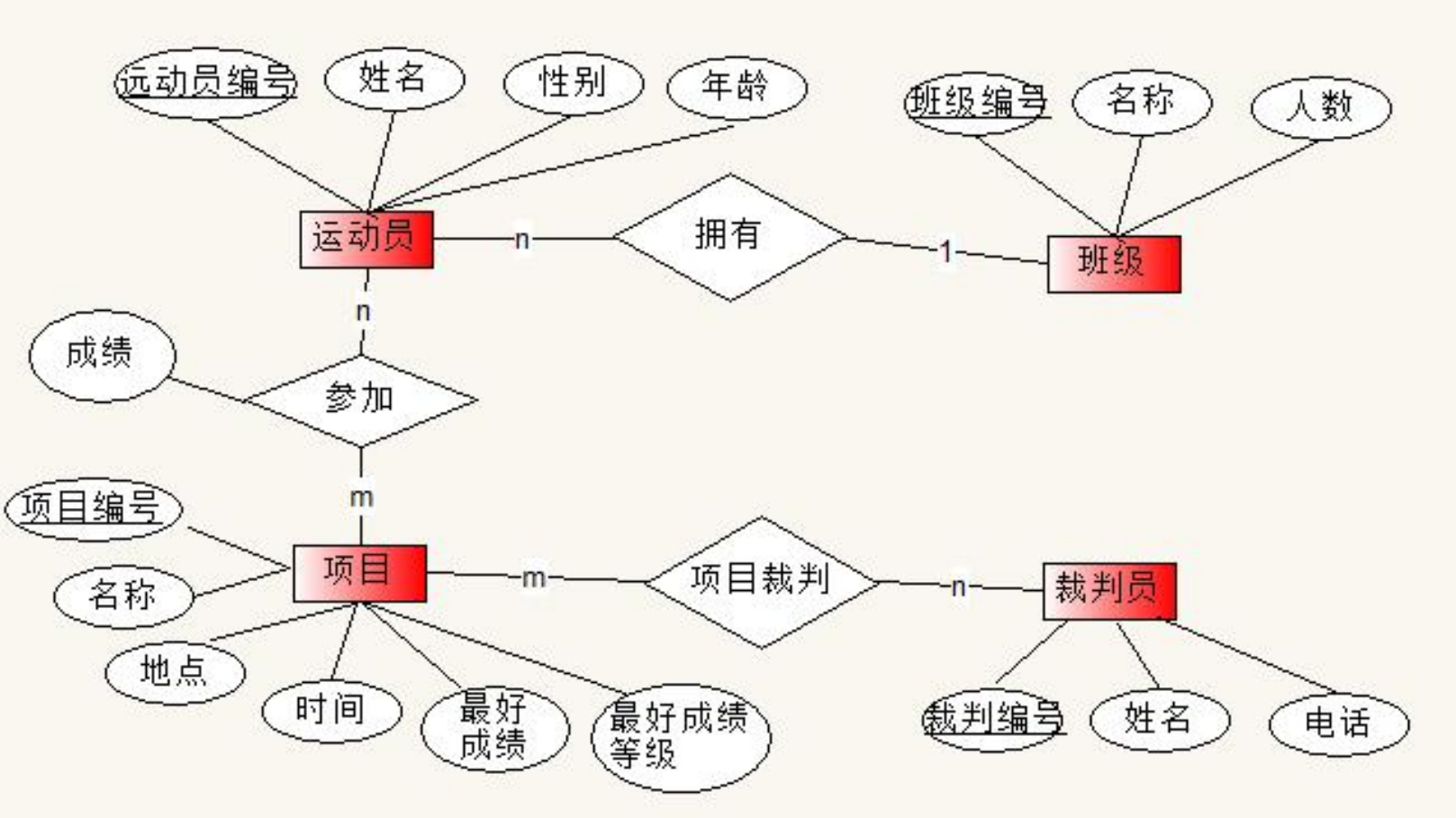
（3）每名裁判员可做多项比赛的裁判，一项比赛需要两个裁判员。系统需记录裁判员号、裁判员姓名、电话等信息。 多对多

此外，系统为了计算运动员个人名次和团队总分，还要记录每个比赛项目的运动员成绩。

联系的属性

6.5 数据库设计综合案例

第二步：画出E-R图



6.5 数据库设计综合案例

第三步：将E-R图转换为关系模型

（1）实体转换

- 班级（班级号，班级名，人数）
- 运动员（运动员号，姓名，性别，年龄）
- 项目（项目编号，名称、比赛地点、比赛时间、最好成绩、最好成绩等级）
- 裁判员（裁判员号，姓名、电话）

（2）联系转换

- 运动员-班级（运动员号，班级号）
- 远动员-项目（运动员号，项目编号，成绩）
- 项目-裁判员（项目编名，裁判员号）

6.5 数据库设计综合案例

第三步：将E-R图转换为关系模型

（3）合并相同码的关系

- 班级（班级号，班级名，人数）
- 运动员（运动员号，姓名，性别，年龄，班级号）
- 项目（项目编号，名称、比赛地点、比赛时间、最好成绩、最好成绩等级）
- 裁判员（裁判员号，姓名、电话）
- 远动员-项目（运动员号，项目编号，成绩）
- 项目-裁判员（项目编名，裁判员号）
- 运动员-班级（运动员号，班级号）

6.5 数据库设计综合案例

第四步：规范化设计

① 班级（班级号，班级名，人数）

② 运动员（运动员号，姓名，性别，年龄 ，班级号 ）

③ 裁判员（裁判员号，姓名、电话）

④ 远动员-项目（运动员号，项目编号，成绩）

⑤ 项目-裁判员（项目编名，裁判员号）

⑥ 项目（项目编号，名称、比赛地点、比赛时间、最好成绩、最好成绩等级）

前5个关系模式都满足BCNF，因为不存在主属性和非主属性对候选码的部分依赖和传递依赖；

6.5 数据库设计综合案例

第四步：规范化设计

项目（项目编号，名称、比赛地点、比赛时间、
最好成绩、最好成绩等级）

最好成绩-→最好成绩等级

所以：项目编号--->最好成绩等级，属于传递依赖

即该关系模式存在非主属性对码的传递依赖，属于2NF

项目（项目编号，名称、比赛地点、比赛时间、最好成绩）

成绩等级（最好成绩，最好成绩等级）

6.5 数据库设计综合案例

关系模式集：

① 班级（班级号，班级名，人数）

② 运动员（运动员号，姓名，性别，年龄 ，班级号 ）

③ 裁判员（裁判员号，姓名、电话）

④ 远动员-项目（运动员号，项目编号，成绩）

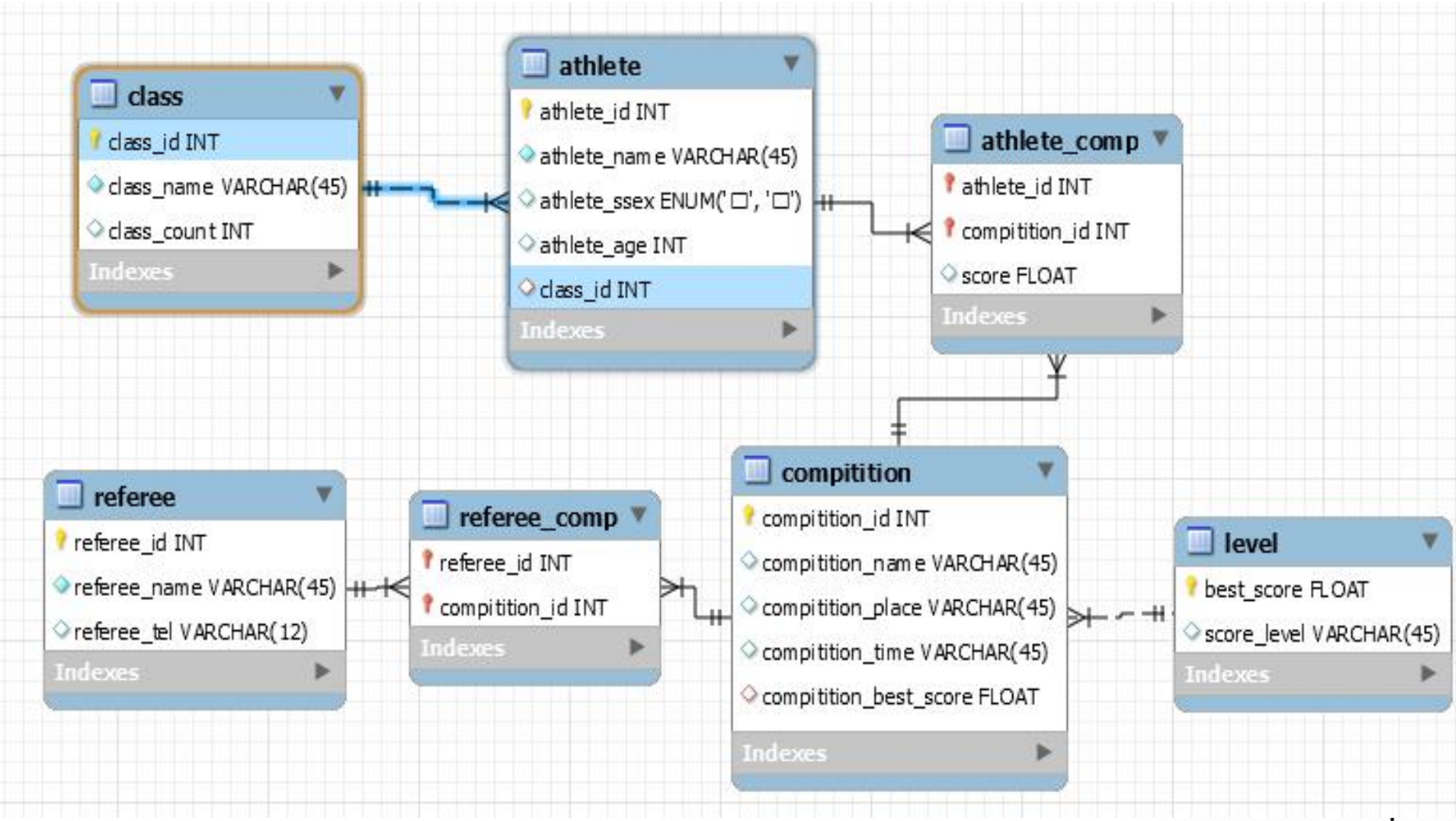
⑤ 项目-裁判员（项目编名，裁判员号）

⑥ 项目（项目编号，名称、比赛地点、比赛时间、最好成绩）

⑦ 成绩等级（最好成绩，最好成绩等级）

6.5 数据库设计综合案例

物理模型

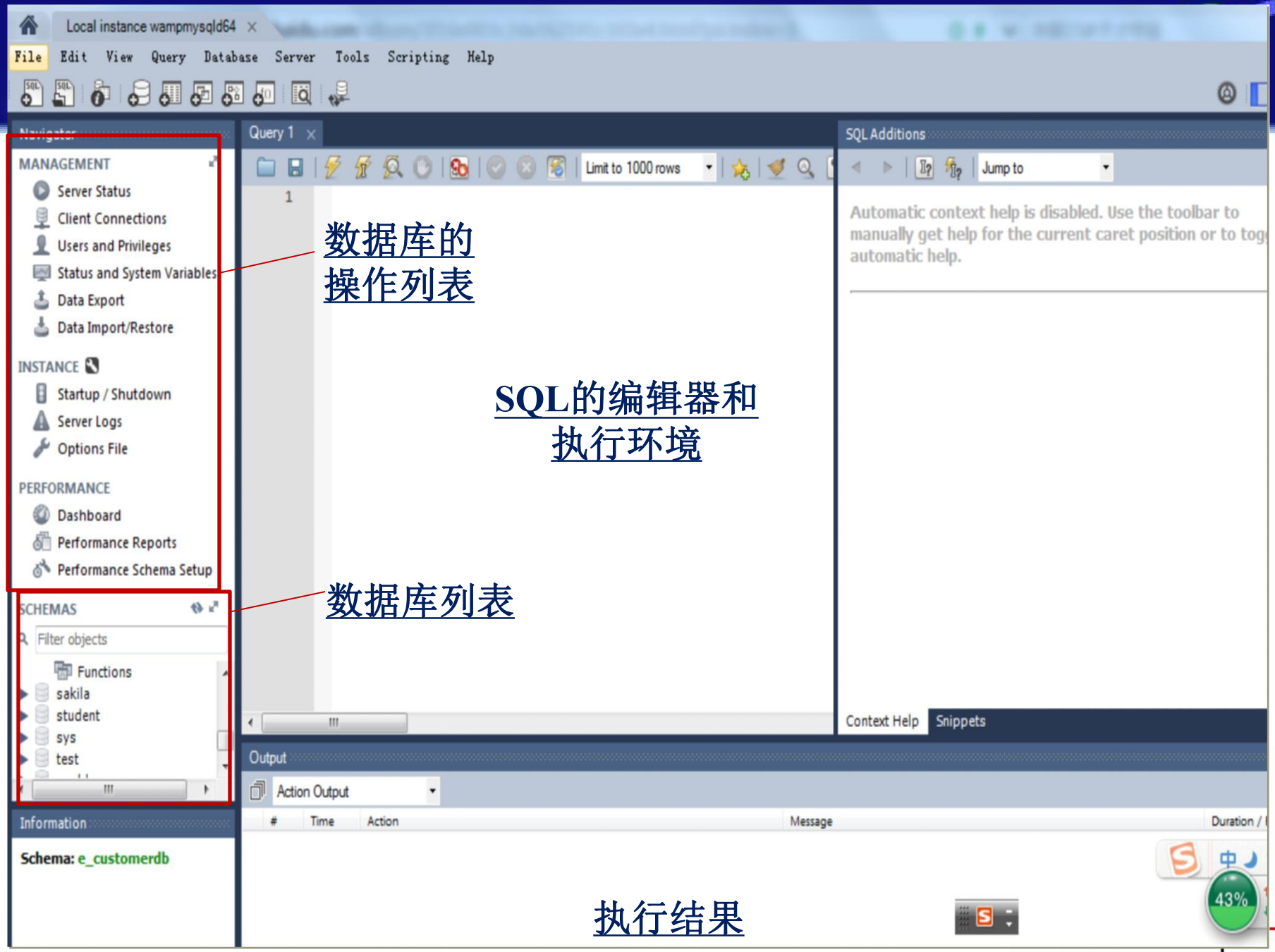


Workbench的使用

MySQL Workbench是一款专为 MySQL设计的数据库建模工具。

可以使用Model生成SQL语句；

反向工程（从库导出ER图）


Local instance wampmysqld64
File Edit View Query Database Server Tools Scripting Help
MANAGEMENT
Server Status
Client Connections
Users and Privileges
Status and System Variables
Data Export
Data Import/Restore
INSTANCE
Startup / Shutdown
Server Logs
Options File
PERFORMANCE
Dashboard
Performance Reports
Performance Schema Setup
SCHEMAS
Filter objects
Functions
sakila
student
sys
test
Information
Schema: e_customerdb
Query 1
Limit to 1000 rows
1
SQL Additions
Jump to
Automatic context help is disabled. Use the toolbar to manually get help for the current caret position or to tog automatic help.
Context Help
Snippets
Output
Action Output
Time Action Message Duration /
数据库的操作列表
SQL的编辑器和执行环境
数据库列表
执行结果

使用workbench的反向工程

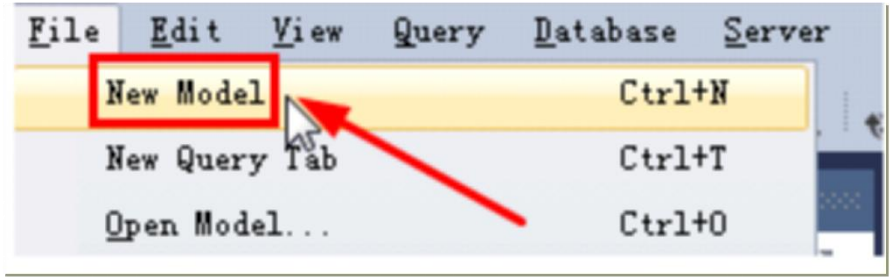
使用workbench从库中导出EER图，即物理模型

Database -> Reverse Engineer，然后一路Next，这期间会让你选择要导出ER图对应的库，最后Finish，反向ER图就生成了。

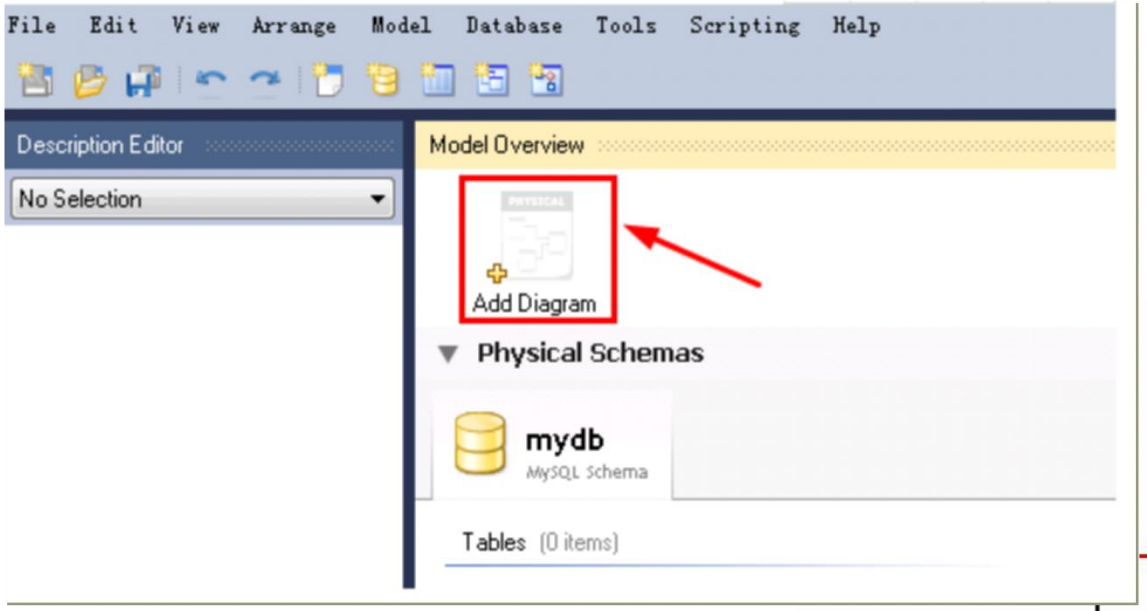
使用workbench创建ER图

使用workbench设计ER图

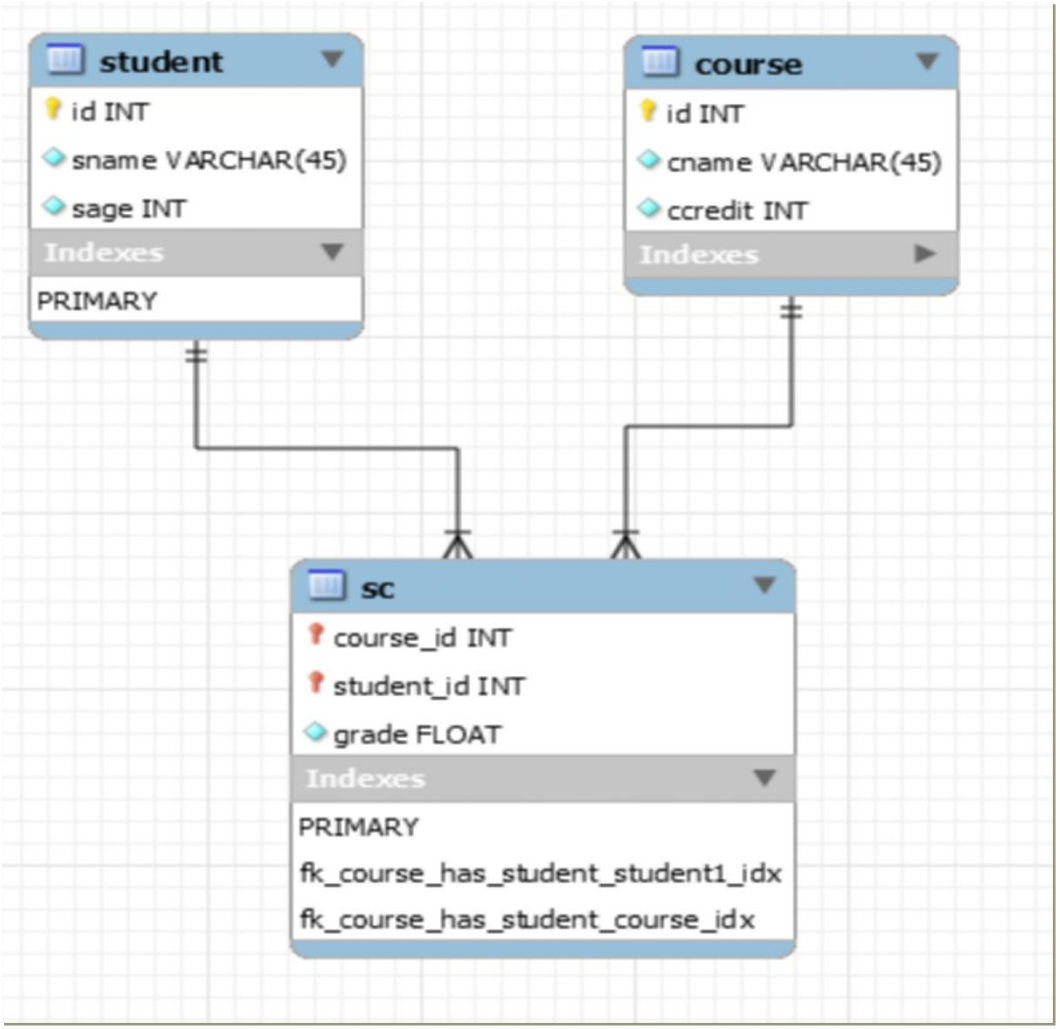
选择File -> New Model：



在新展开的页面中“Model Overview”界面双击“Add Diagram”图标：

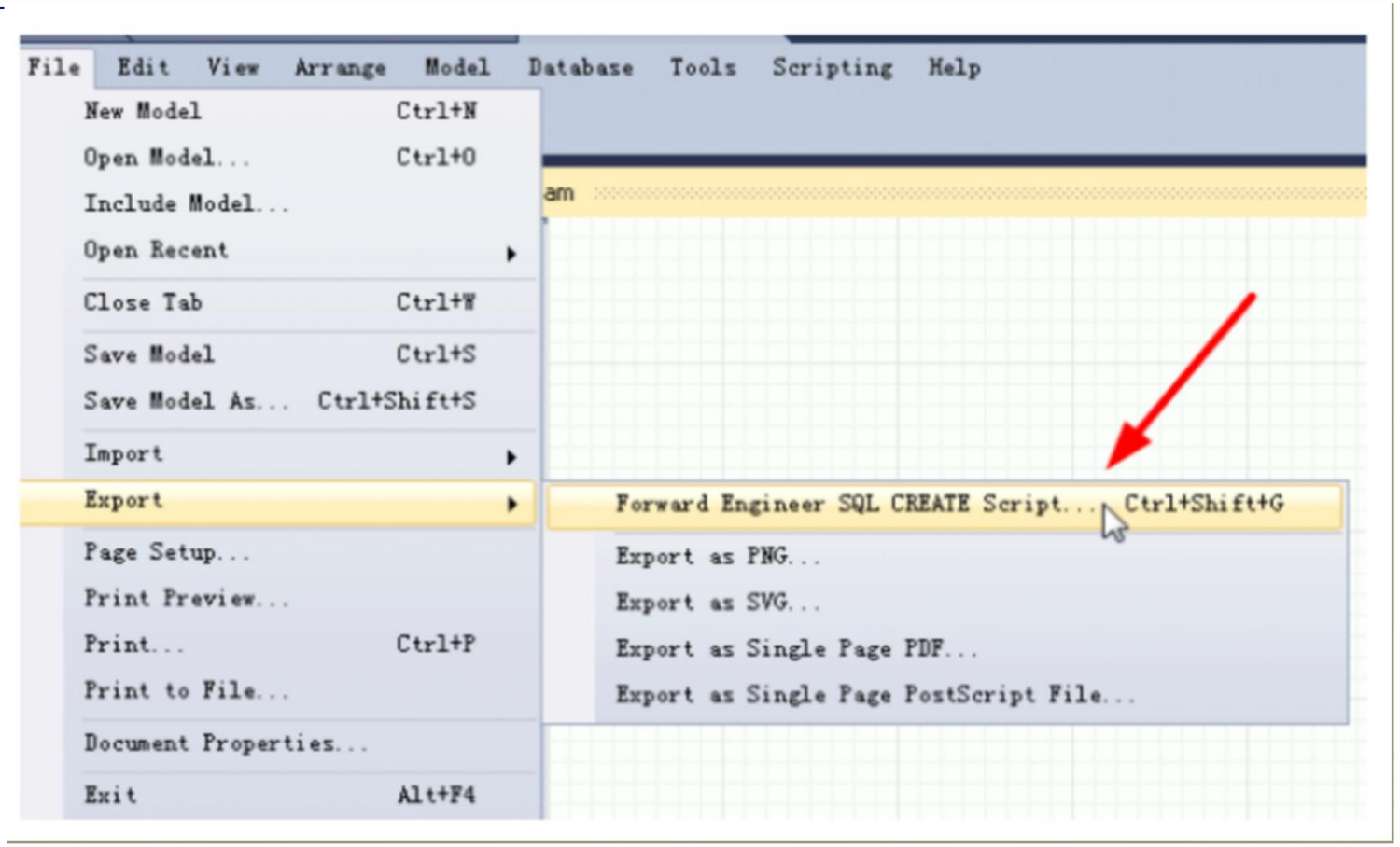


使用workbench创建ER图



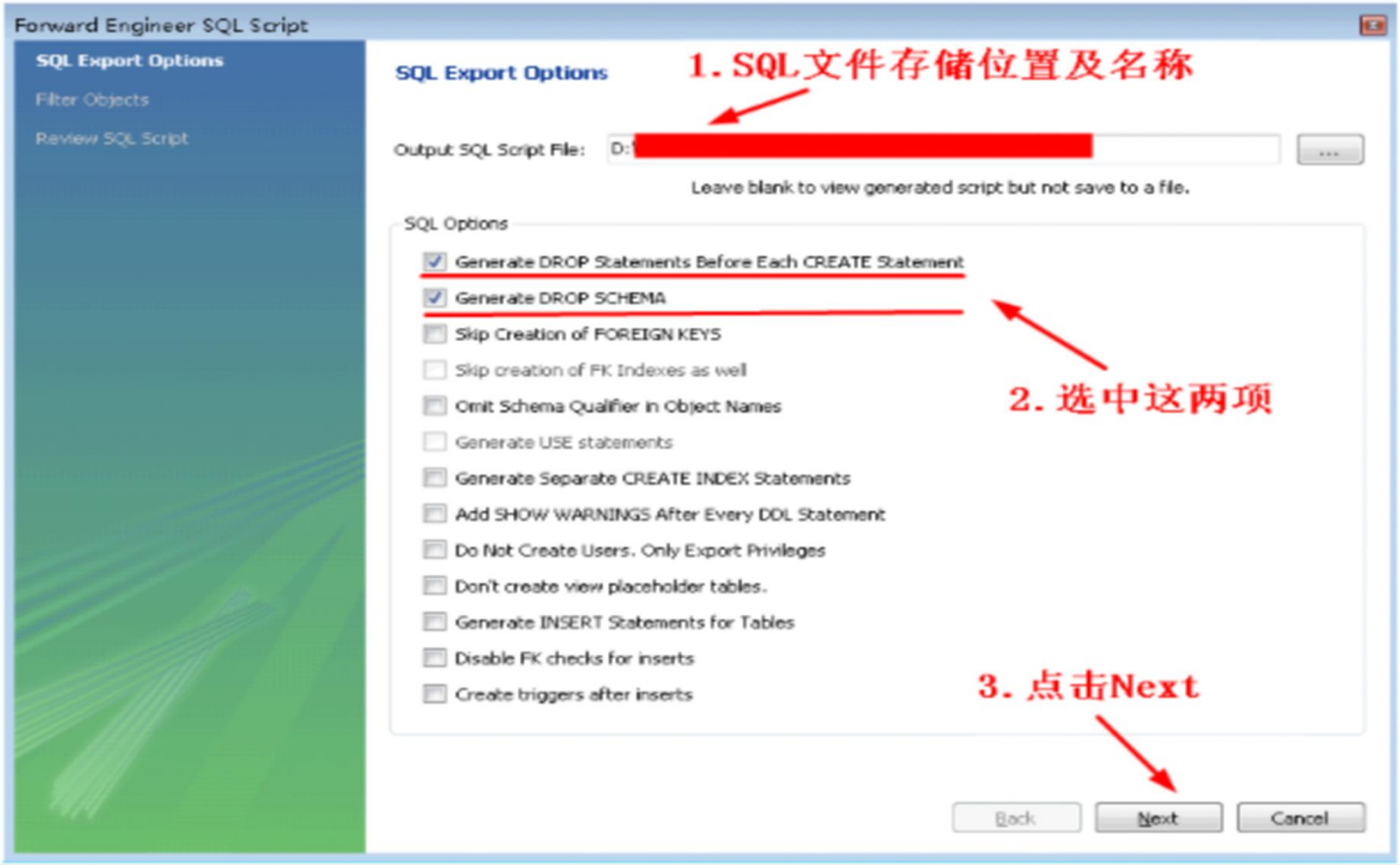
使用workbench生成SQL语句

需要点击File -> Export -> Forward Engineer SQL CREATE SCRIPT

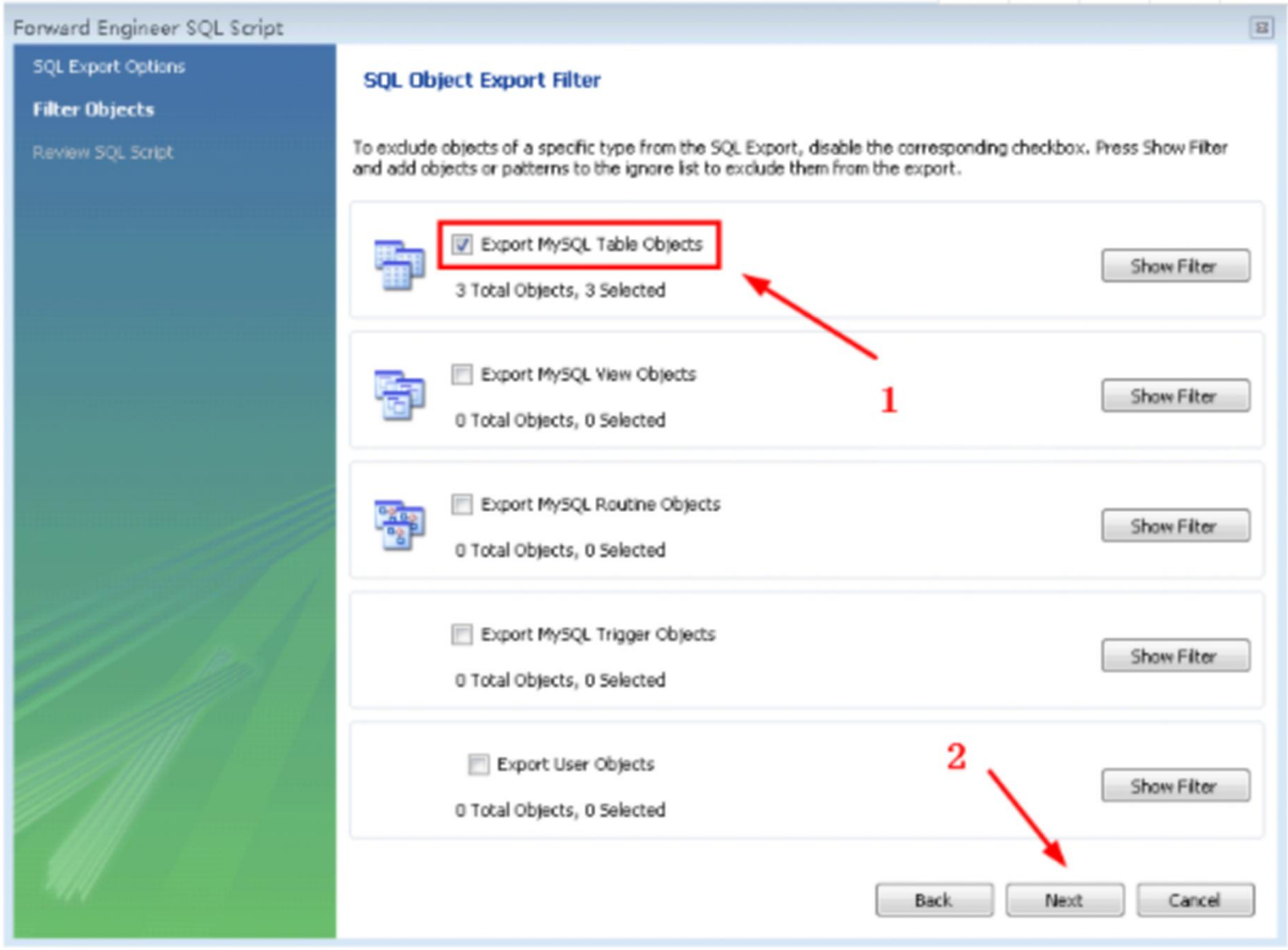


使用workbench生成SQL语句

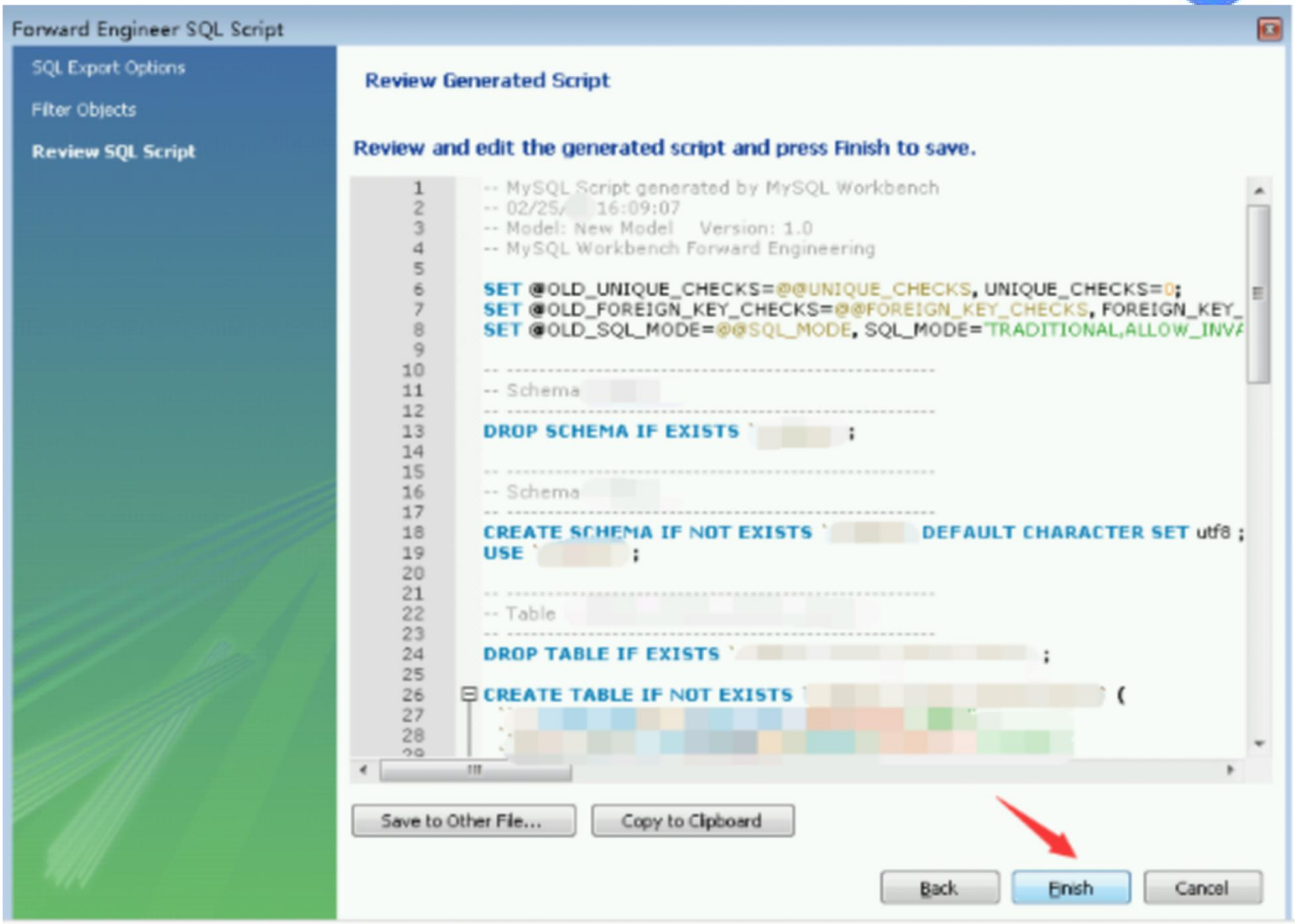
打开“Forward Engineer SQL SCRIPT”，如图：



使用workbench生成SQL语句



使用workbench生成SQL语句

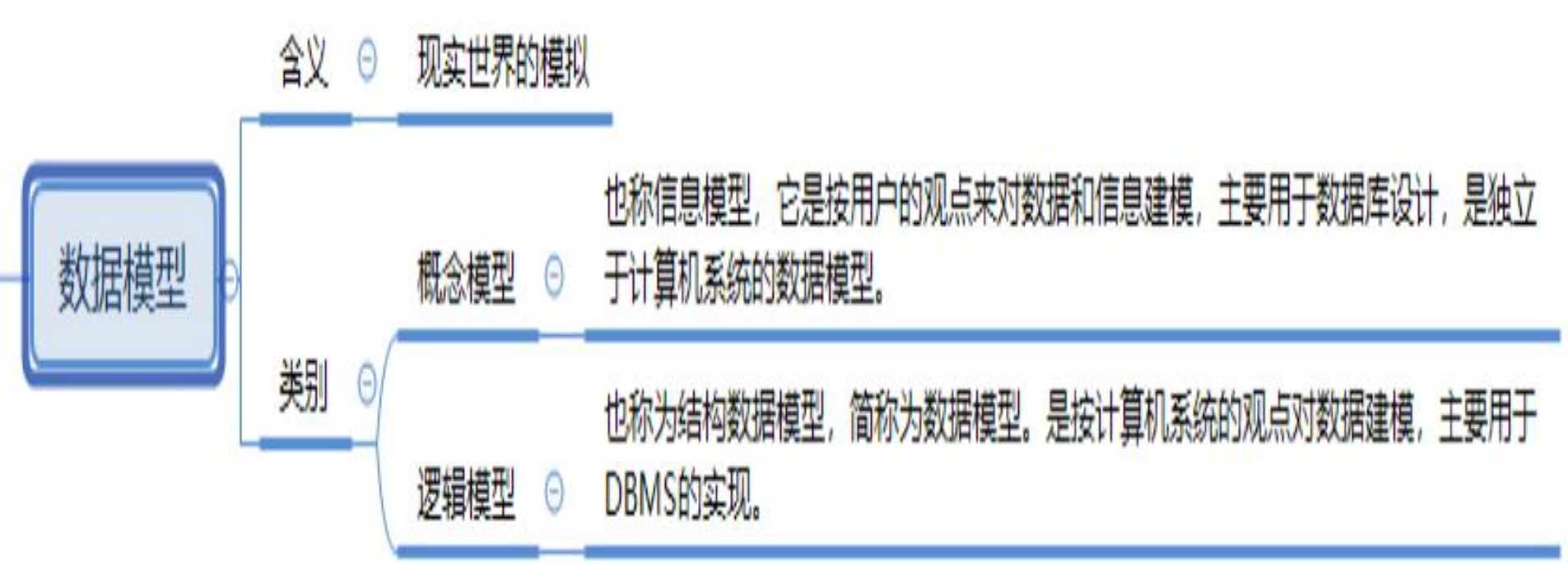


小结

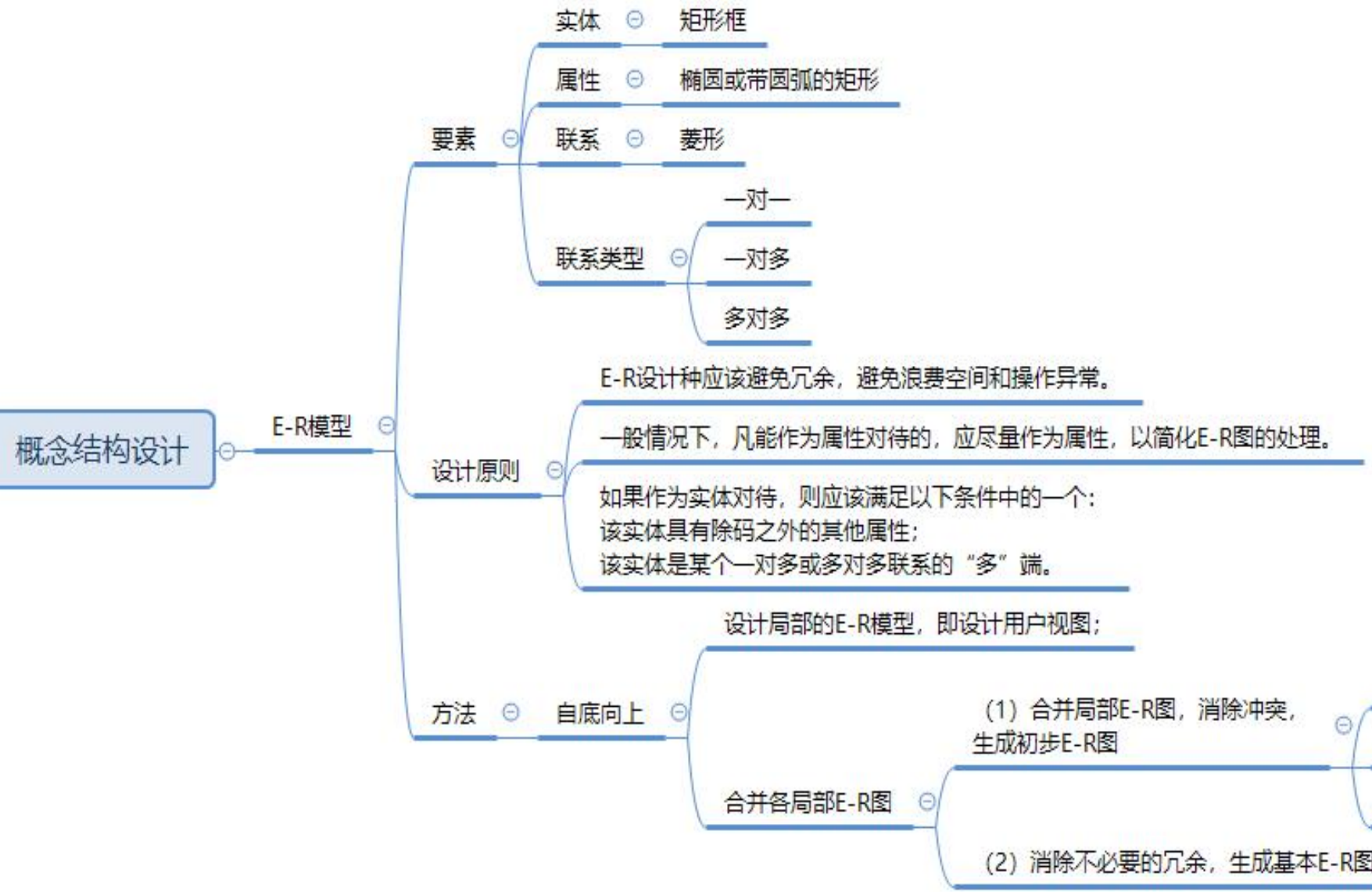
小结


数据库设计概述
任务
根据一个应用环境的信息需求和处理需求，
以及数据库所需的DBMS、操作系统和硬件的特性
设计出最优的数据库模式和应用程序
特点
反复性
多解性
分步进行
结构设计和行为设计相结合
方法
E-R模型设计法
新奥尔良法
3NF设计法
基于视图的设计法
步骤
规划：可行性分析
需求分析
概念结构设计
逻辑结构设计
物理结构设计
编码实现
运行与维护

小结

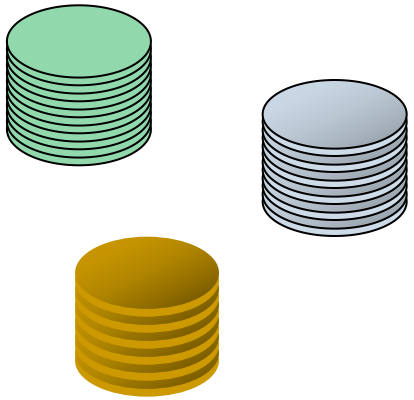

数据模型
含义
现实世界的模拟
类别
概念模型
也称信息模型，它是按用户的观点来对数据和信息建模，主要用于数据库设计，是独立于计算机系统的数据模型。
逻辑模型
也称为结构数据模型，简称为数据模型。是按计算机系统的观点对数据建模，主要用于DBMS的实现。

小结


概念结构设计
E-R模型
要素
实体
矩形框
属性
椭圆或带圆弧的矩形
联系
菱形
联系类型
一对一
一对多
多对多
设计原则
E-R设计种应该避免冗余，避免浪费空间和操作异常。
一般情况下，凡能作为属性对待的，应尽量作为属性，以简化E-R图的处理。
如果作为实体对待，则应该满足以下条件中的一个：
该实体具有除码之外的其他属性；
该实体是某个一对多或多对多联系的“多”端。
方法
自底向上
设计局部的E-R模型，即设计用户视图；
合并各局部E-R图
(1) 合并局部E-R图，消除冲突，生成初步E-R图
属性冲突
命名冲突
结构冲突
(2) 消除不必要的冗余，生成基本E-R图；

小结


逻辑结构设计
步骤
将E-R图转换成关系模式
实体转换
关系的属性：实体的属性
关系的码：实体的码
多对多联系转换
关系的属性：各端码+自身的属性
关系的码：各端的码
一对多联系转换
关系的属性：原关系的属性+联系自身的属性+1端的码
关系的码：不变
一对一联系转换
关系的属性：原关系的属性+联系自身的属性+另一1端的码
关系的码：不变
规范化处理
满足3NF或BCNF
设计用户外模式（子模式）



数据库系统

第七章 视图和索引

北京工业大学耿丹学院
计算机科学与技术专业

内容导航


视图
引入视图的目的
使用SQL命令创建和删除视图
利用视图更新、插入和删除记录
索引
引入索引的目的
索引的类型
创建索引
删除索引

什么是视图?

	学号	姓名	性别	年龄	身份证号	班级编号
1	31022001	曹操	男	27	111111111111111001	310220
2	31022002	司马懿	男	24		
3	31022003	蔡文姬	女	27		
4	31031001	刘备	男	26		
5	31031002	诸葛亮	男	25		

	学号	课程号	成绩	学期
1	31022001	30001	88	1
2	31022001	30002	80	2
3	31022001	30003	95	3
4	31022001	30004	85	4
5	3			4

基于学生基本情况表表和成绩表创建视图

	学号	姓名	性别	课程号	成绩
1	31022001	曹操	男	3000	
2	31022001	曹操	男	3000	
3				3000	
4				3000	
5				3000	
6	31022002	司马懿	男	3000	
7	31022002	司马懿	男	3000	

教师需要的视图：
方便查看学生的成绩

	学号	姓名	性别	年龄
1	31022001	曹操	男	27
2	31022002	司马懿	男	24
3	3102			
4	3103			
5	31031002	诸葛亮	男	25

班主任需要的视图：
方便查看学生的档案

视图的定义

- 视图是一张虚拟表，它表示一张表的部分数据或多张表的综合数据，其结构和数据是建立在对表的查询基础上
- 同一张原始表，根据不同用户的不同需求，可以创建不同的视图
- 优点：
- 简化查询语句

日常开发中我们可以将经常使用的查询定义为视图，从而使用户避免大量重复的操作。

- 安全性

通过视图用户只能查询和修改他们所能见到的数据，数据库中的其他数据则既看不到也取不到。

逻辑数据独立性

- 视图可以帮助用户屏蔽真实表结构变化带来的影响。

创建视图

- 使用SQL语句创建视图的语法

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

- 说明：

- (<列名> [，<列名>]…)为可选项，省略时，视图的列名由子查询的结果决定。

- WITH CHECK OPTION 通过视图进行增删改操作时，不得破坏视图定义中的谓词条件（即子查询中的条件表达式）

创建视图

- 常见的视图形式

行列子集视图
With check option 的视图
基于多个基表的视图
基于视图的视图
带表达式的视图
分组视图

行列子集视图

示例1

在学生选课数据库中,根据现有的学生表（S）创建一个仅包含“IS”的所有学生的视图并查看该视图。

示例1答案

```
create or replace view  view_IS_S
as
Select  *   from  s where Sdept='IS';

Select * from view_IS_S
```

行列子集视图

示例2

在学生基本情况表（s）上的建立包含学号，姓名，年龄的所有男生的视图，

示例2答案

```
create view view_S1
as
Select  sno,sname,sage
from S
where Ssex= '男'
```

带 WITH CHECK OPTION 的视图

规定对视图所执行的所有数据执行修改操作时都必须遵守视图定义中select语句所设置的条件

示例3

建立所有男生学生的视图，并要求通过该视图进行的更新操作只涉及男生。

示例3答案

```
create view view_S2
as
select Sno,Sname,sse
where Ssex= '男'
with check option
```

思考：下列语句是否可以执行？

1）update view_S2
　　set ssex='女‘

2）insert into view_S2
　　values('8','A','男',18,'IS')

基于多个基本表的视图

示例4

创建视图View_SC,包括计算机系各学生及其选修课程的情况，要求对该视图的修改都符合系别为“CS”这一条件。

示例4答案

```
CREATE VIEW View_SC AS
SELECT s.*,cno,grade
FROM S,SC
WHERE S.sno=SC.sno and Sdept= 'CS'
WITH CHECK OPTION;
```

基于视图的视图

示例5

创建计算机系选修了课程号为“1”的所有学生的选课情况的视图View_S。

示例5答案

```
CREATE VIEW View_S AS
 SELECT Sno,Sname,Cno,Grade
 FROM View_SC
 WHERE cno=1
```

带表达式的视图

示例6

对于学生选课数据库，创建一个查询计算“IS”系学生人数的视图。

示例6答案

```
create view  view_IS_s_count (人数)
as
select count(*)
from s
where sdept='IS'
```

分组视图

示例7

对于学生选课数据库，创建一个查询每名学生平均成绩的视图。

示例7答案

```
create view view_s_average_score(sno,score)
as
select sno, avg(grade)
from sc
group by sno
```

如果是sql server必须给列avg(score)指定列名

删除视图

删除视图的语法结构

```
drop view <视图名>  [, …n]
```

示例8

删除以上所创建的视图
view_s_average_score、view_IS_s_count。

示例8答案

```
drop view view_s_average_score,
view_IS_s_count
```

通过视图更新记录

示例9

通过视图view_sc对表（s）和（SC）作如下修改：
（1）将姓名为李勇且课程号为“1”的成绩更新为78分。
（2）将姓名为李勇且课程号为“1”的成绩更新为88分，性别更新为“女”。

示例9答案

（1）
```
update view_sc  set grade=78
    where sname='李勇' and cno='1‘
```
（2）
```
update view_sc set grade=88,ssex=’女’
    where sname= '李勇' and   cno='1‘
```

注：（2）修改基表不成功的原因是不能同时对两个基表修改。

通过视图更新记录

通过视图view_sc对表（s）和（SC）作如下修改：

（1）将姓名为李勇且课程号为“1”的成绩更新为78分。

（2）将姓名为李勇且课程号为“1”的成绩更新为88分，性别更新为“女”。

（1）update view_sc set grade=78
where sname='李勇' and cno='1‘

（2）update view_sc set grade=88,ssex=’女’
where sname= '李勇' and cno='1‘

```
update view_sc set grade=88
where sname= '李勇' and   cno='1'
update view_sc set ssex='女'
where sname= '李勇' and   cno='1'
```

通过视图插入记录

示例10

（1）对（s）创建一个新视图v_s1。
（2）通过视图v_s1向s表中插入一条新记录。

示例10答案

```
（1）Create view v_s1
     As Select * from s
（2）insert into     v_s1
      values (‘10','刘表','男',‘20',‘IS')
```

通过视图删除记录

示例11

（1）通过视图v_s1按如下删除s表一条新记录。

```
delete from v_s1
where sname='刘表'
```

示例12

（2）通过视图view_sc中按如下语句删除（s）一条记录。

```
delete from view_sc        where sname='刘晨'
```

注：删除数据不成功的原因是不能同时对两个基表数据进行删除。

查询视图

- 从用户角度：查询视图与查询基本表相同
- DBMS实现视图查询的方法

  视图消解法（View Resolution）
  - ✓ 进行有效性检查，检查查询的表、视图等是否存在。如果存在，则从数据字典中取出视图的定义。
  - ✓ 把视图定义中的子查询与用户的查询结合起来，转换成等价的对基本表的查询。
  - ✓ 执行修正后的查询。

示例13

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    View_IS_S
WHERE   Sage<20;
```

View_IS_S视图的定义 (视图定义例1)：

```
CREATE VIEW View_IS_S
```

用视图消解法，转换后的查询语句为：

```
SELECT  Sno，Sage
FROM  S
WHERE  Sdept= 'IS'  AND  Sage<20;
```

示例14

查询信息系统选修了1号课程的学生

```
SELECT  Sno，Sname
FROM    View_IS_S，SC
WHERE  View_IS_S.Sno =SC.Sno AND
SC.Cno= '1';
```

视图可以简化SQL程序设计，同时也可看做是数据库的一个安全措施

索引

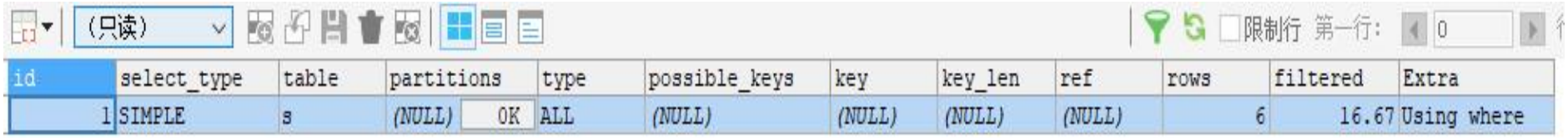
- Select * from s where sno=1000
- Mysql数据库必须从第1条开始遍历，直到找到sno=1000的数据
- 效率显然很低
- MYSQL允许建立索引加快数据表的查询和排序。

什么是索引

- 汉语字典中的汉字按页存放，一般都有汉语拼音目录（索引）、偏旁部首目录等
- 我们可以根据拼音或偏旁部首，快速查找某个字词

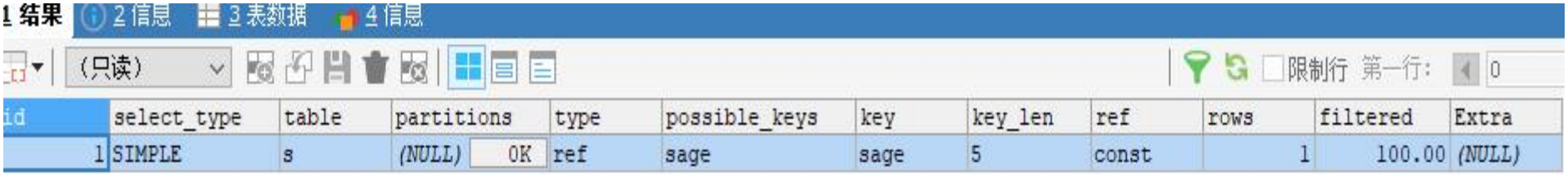
- 在数据库操作中，经常需要查找特定的数据，例如：
- **select * from s where sage=20;**

explain select * from s where sage=20;



id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	SIMPLE	s	(NULL)	ALL	(NULL)	(NULL)	(NULL)	(NULL)	6	16.67	Using where

```
# 创建索引
 create index sage on s(sage);
#再次执行
  explain select * from s where sage=20;
```



id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	SIMPLE	s	(NULL)	ref	sage	sage	5	const	1	100.00	(NULL)

索引类型

- 普通索引：由key 或index定义的索引
- 唯一索引：由unique定义的索引，唯一索引不允许两行具有相同的索引值。
- 单列索引：在表中单个字段上创建索引。
- 多列索引：在表中多个字段上创建索引。需要注意，只有在查询条件中使用了这些字段中的第一个字段时，该索引才会被使用。

SQL语句命令创建索引

- 创建索引的语法：

1.创建表的时候创建索引

CREATE table 表名（字段名 数据类型 完整性约束，字段名 数据类型 完整性约束,...
[unique] INDEX|KEY [索引名]（字段名）
[ASC|DESC])；
asc或desc指定升序或降序的索引值存储

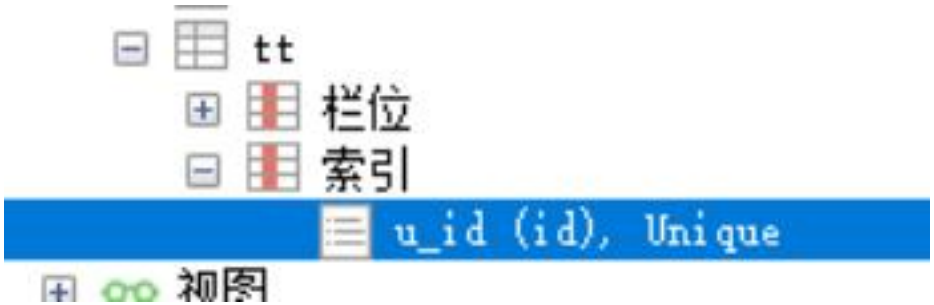
例1.创建普通索引

```
create table t2(
    id int primary key,
    name CHAR(20),
    sage int,
    index t2_name(name)
);
```

例2：创建唯一索引

创建一个唯一索引，

```
create table tt(
id int not null,
sname varchar(20),
sage int,
unique index u_id(id asc));
```



例3：创建多字段索引

```
create table tt1(
id int not null,
sname varchar(20),
sage int,
index (id ,sname),
index (id));
```

```
176  show create table tt1;
```

1 结果 | 2 信息 | 3 表数据 | 4 信息

(只读)

Table	Create Table
tt1	CREATE TABLE `tt1` (`id` int(11) NOT NULL, `sname` varchar(20) DEFAULT NULL, `sage` int(11) DEFAULT NULL, KEY `id` (`id`,`sname`), KEY `id_2` (`id`)) ENGINE=InnoDB DEFAULT CHARSET=utf8

如果不加索引名，那么MySQL会以索引的第一个字段的名字来命名

而如果一个表下有多个索引的第一个字段都是相同的，那么索引名会在字段名后加序数

例4：

多列索引：在表中多个字段上创建索引。需要注意，只有在查询条件中使用了这些字段中的第一个字段时，该索引才会被使用。

我们向表插入一条数据，然后使用**EXPLAIN**语句查看索引是否有在使用

```
insert into tt1
values(1,'a',17),(2,'aa',19),
(3,'ab',16),(4,'ac',12),(5,'b',13),(6,'c',16);
```

```
186   explain select * from tt1 where id=5;
```

1 结果 | 2 信息 | 3 表数据 | 4 信息

（只读） 限制行 第一行:

id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	SIMPLE	tt1	(NULL)	ref	id,id_2	id	4	const	1	100.00	(NULL)

```
188   explain select * from tt1 where sname like 'a%';
```

1 结果 | 2 信息 | 3 表数据 | 4 信息

（只读） 限制行 第一行:

id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	SIMPLE	tt1	(NULL)	ALL	(NULL)	(NULL)	(NULL)	(NULL)	6	16.67	Using where

SQL语句命令创建索引

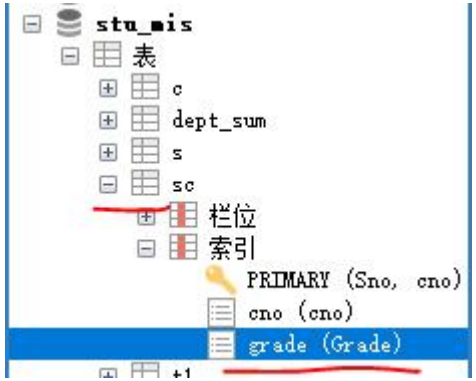
2.在已有的表的基础上创建索引

(1)create index 索引名 on 表名（字段名[asc|desc]）；
(2) alter table 表名 add index 索引名（字段名[asc|desc]）

例如：在SC 表上对成绩建立普通索引

```
create index grade on sc(grade);
```

```
alter table sc add index grade(grade);
```



利用SQL语句删除索引

```
drop  index  索引名  on 表名
```

示例

删除学生学生选课表中已有的索引:grade

示例答案

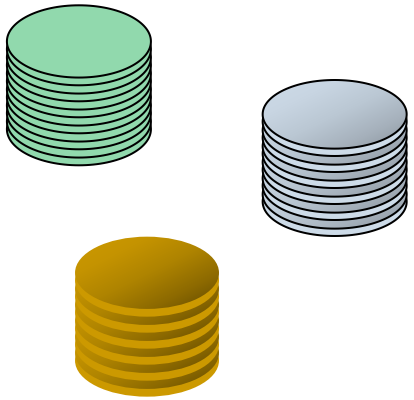
```
drop index grade on sc;
```

索引的优缺点

- 优点
  - 加快访问速度
  - 加强行的唯一性
- 缺点
  - 带索引的表在数据库中需要更多的存储空间
  - 操纵数据的命令需要更长的处理时间，因为它们需要对索引进行更新

创建索引的指导原则

- 请按照下列标准选择建立索引的列。
  - 该列用于频繁搜索
  - 该列用于对数据进行排序
- 请不要使用下面的列创建索引：
  - 列中仅包含几个不同的值。
  - 表中仅包含几行。为小型表创建索引可能不太划算，索引中搜索数据所花的时间比在表中逐行搜索所花的时间更长



数据库系统

第八章 存储过程、触发器、事务

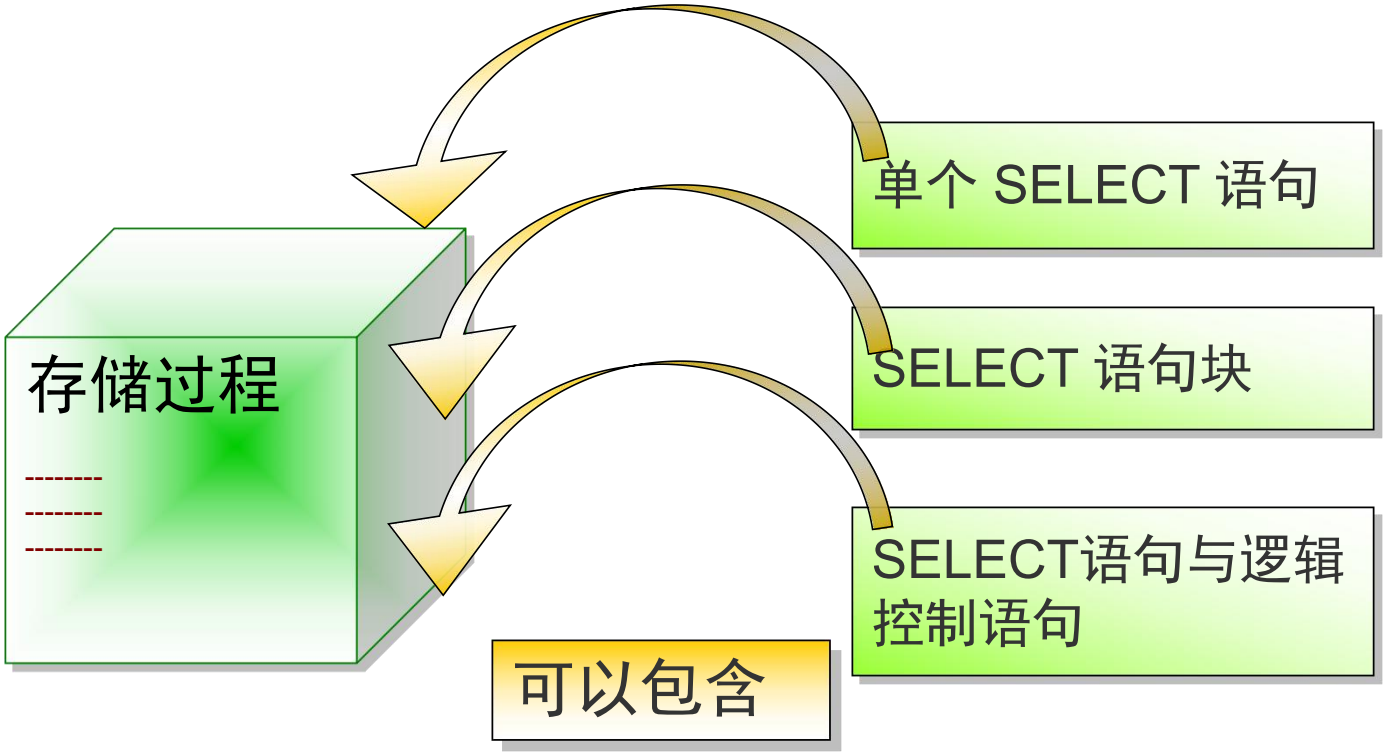
北京工业大学耿丹学院
计算机科学与技术专业

存储过程、触发器、事务

1. 存储过程的创建、调用基本操作
2. 触发器基本操作
3. 事务的概念、特性、开启、提交、回滚

什么是存储过程

- 存储过程可以包含数据操纵语句、变量、逻辑控制语句等



1 存储过程

- 存储过程是将SQL语句放到一个集合里，然后直接调用存储过程来执行已经定义好的SQL语句集合。
- 可以避免开发人员重复编写相同的SQL语句。
- 可以减少数据在数据库和应用服务器之间的传输，提高数据的处理效率

创建存储过程

- 存储过程示例1—返回单个数据
- 统计一共有多少名学生

```
select COUNT(*) from s;
```

```
delimiter $$
create procedure s_num(out num int)
begin
   select COUNT(*) into num from s;
end;$$
```

delimiter $$将语句的结束符号从分号;临时改为两个$$(可以是自定义)

调用函数call

```
delimiter;
call s_num(@a);
select @a;
```

变量

- **Mysql中变量不需要声明，可以直接使用@**

```
#第一种用法：
set @num=1;
select @num;
set @num:=100;
```

```
#第二种用法
select @num:=1000 b;
select @num:=COUNT(*) 学生总数 from s;
select COUNT(*) into @num from s;
select @num;
```

注意：
使用set时可以用“=”或“：=”，
但是使用select时必须用“：=赋值”

- Mysql中变量的声明，declare可以直接使用,只能用在存储过程中

变量的声明declare

```
-- 无效
declare a int;
set a=1;
```

```
-- declare声明需要放在存储过程中
drop procedure if exists p_s;
delimiter//
create procedure p_s()
begin
 declare a int;
 set a=100;
 select a;
end;//
```

```
drop procedure if exists p_s1;
delimiter//
create procedure p_s1()
begin
 -- declare a int;
 set @a=100;
 select @a;
end;//
```

```
-- 调用存储过程
call p_s1;
call p_s;
```

创建存储过程

❖存储过程示例2—返回数据集

```
CREATE PROCEDURE getrecord()
BEGIN
   SELECT * FROM s;
END;
```

```
delimiter//
create procedure getrecord()
begin
  select * from s;
end;//
```

❖调用

```
call getrecord();
```

创建存储过程的语法结构

- 定义存储过程的语法结构

CREATE PROCEDUR *sp_name* ([*proc_parameter*[,...]])

过程体

注意：即使没有参数，()也不能省略。

- ***proc_parameter***: **[IN | OUT | INOUT]** ***param_name type***

- ***in: 表示该参数的值必须在调用存储过程时指出***
- ***Out:表示该参数的值可以被存储过程改变，并且可以返回***
- ***Inout:表示该参数调用时需指出，并且可以被改变和返回***

调用存储过程

- call 语句用来调用存储过程
- 调用的语法

  call 过程名 [参数]

```
call getrecord();
```

简单存储过程的创建

示例1

创建一个简单的存储过程，输出hello world，并执行该存储过程。

示例1答案

```
Delimiter //
create  procedure  proc()
BEGIN
        select 'hello world';
END;//
```

调用存储过程

```
call proc();
```

带有输入参数的存储过程的创建

示例2

创建一个带输入参数的简单存储过程：计算某个学生所选课程的总数，并执行该存储过程。

示例2答案

```
delimiter //
create procedure sp_sc_count(in id int)
begin
  select COUNT(cno) from sc where sno=id;
end;//
```

调用存储过程

```
call sp_sc_count(1);
```

带有输入参数的存储过程的创建

练习：创建一个测试表test1(id int,sname varchar(10)),创建一个含有两个输入参数的存储过程，用来给test1表插入数据

```
create table test1(
id int,sname varchar(10));

delimiter //
create procedure sp_test_insert(in id int,in name varchar(10))
begin
  insert into test1 values(id,name);
end;


-- 调用
call sp_test_insert(1,'A');
call sp_test_insert(2,'B');
select * from test1;
```

带有一个输入和一个输出参数的存储过程

示例3

统计某课程的选课人数，
#创建带有一个输入参数和一个输出参数的存储过程。
#并执行该存储过程。

示例3答案

```
delimiter//
create procedure sp_c_count(in id int,out scount int)
begin
   select COUNT(sno)into scount from sc where cno=id;
end;//
```

执行带有输出参数的存储过程

- 在调用带有输出参数的存储过程时

  输出参数必须是一个带@符号的变量

```
call sp_c_count(1,@num);
select @cno:=1 课程号,@num 选课人数;
```

```
delimiter//
create procedure sp_c_count(in cid int,out scount int)
begin
select COUNT(sno) into scount from sc where cno=cid;
select cid, scount;
end;//
call sp_c_count(1,@scount);
```

带有一个参数，既是输入参数又是输出参数的存储过程

示例4

传一个数值，让它自动增加10
#创建一个带有既是输入参数又是输出参数的存储过程。
#并执行该存储过程。

示例4答案

```
drop procedure if exists p_3;
delimiter//
create procedure p_3(inout m int)
begin
   set m=m+10;
   select m;
end;//
```

调用存储过程

```
set @m=3;
call p_3(@m);
```

调用的时候，inout型的参数值既是输入类型又是输出类型，给它一个值，值不是变量，因此我们需要先设置一个变量并初始化这个值，调用的时候直接传这个变量即可。

存储过程课堂练习

练习1

统计学生表中男女总数，创建一个不带有任何参数的存储过程

练习1答案

```
delimiter//
create procedure sp_ssex_count()
begin
select ssex,COUNT(sno)人数 from s group by ssex;
end;//
```

调用存储过程

```
call sp_ssex_count();
```

删除存储过程

- 删除存储过程语法结构：

```
DROP  Procedure  <存储过程名>;
```

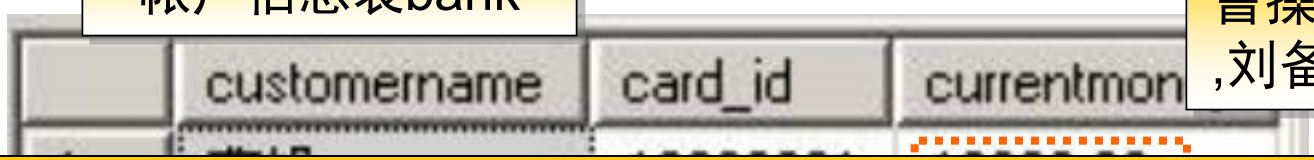
```
drop  procedure sp_ssex_count;
```

为什么需要触发器

- 为什么需要触发器(TRIGGER)呢？典型的应用就是银行的取款机系统

帐户信息表bank

曹操开户10000元
,刘备开户1元



额

最优的解决方案就是采用触发器：

- 它是一种特殊的存储过程
- 也具备事务的功能
- 它能在多表之间执行特殊的业务规则

什么是触发器

触发器（Trigger）是用户定义在关系表上的一类由事件驱动(insert、update、delete)的特殊过程。利用触发器能够有效地保证数据完整性，也便于执行一些自动的数据操作。

触发器和存储过程的区别

- 存储过程的定义可有参数，触发器的定义不能有参数
- 执行方式不同，触发器由引起表数据变化的操作（增insert、删delete、修update）触发而自动执行，存储过程必须通过具体的语句调用

创建触发器基本语法

- 触发器创建的基本语法：

CREATE TRIGGER 触发器名 触发时间 触发事件

ON 表名 FOR EACH ROW 触发器语句

- 触发器只能建立在基本表上，不能建立在临时表或视图上。
- 触发时间：before或after;before是检查约束前触发；after是检查约束后触发。
- 触发事件包括：INSERT、UPDATE、DELETE
- FOR EACH ROW行级触发器，MYSQL目前还没有支持语句级的触发器。
- 一个表上不能有两个相同时间和事件的触发程序。

创建触发器基本语法

- 使用OLD(旧值)和NEW(新值)关键字，能够访问受触发程序影响的行中的字段值（OLD和NEW不区分大小写）。
- 在INSERT触发程序中，仅能使用NEW.col_name，没有旧值。
- 在DELETE触发程序中，仅能使用OLD.col_name，没有新值。
- 在UPDATE触发程序中，可以使用OLD.col_name来引用更新前的旧值，也能使用NEW.col_name来引用更新后的行中的新值。

- before：(insert、update)可以对new进行修改，
- after:不能对new进行修改；
- 两者都不能修改old数据。

```
-- 创建一个触发器，只要向s表中插入一条记录，记在日志表中插入当前时间
create table s_log(
id int primary key auto_increment,
t_diary datetime)

create trigger tri_s before insert on s
for each row
insert into s_log(t_diary)value(NOW());

insert into s(sname) values('C');

select * from s_log;
```

触发器举例，实现级联更新

利用触发器实现学生与成绩表学生编号上的级联更新。

```
create trigger tri_s_update_sc after update
on s
for each row
update sc set sno=new.sno where sno=old.sno;
```

```
-- 验证
  update s set sno=17001 where sno=1;
-- 如果发生错误
  set foreign_key_checks=0; /*关闭外键约束*/
  update s set sno=17001 where sno=1;
  select * from s;
  select * from sc;
```

触发器实现级联删除

利用触发器实现学生与成绩表学生编号上的级联删除。

```
create trigger tri_s_delete_sc before delete on s
for each row
delete from sc where sno=old.sno;
```

验证

```
delete from s where sno=2;
```

练习

- 在学生基本情况表（s）上建立一个插入触发器，实现当向表中插入一条记录时系表(sdept)相应的系人数自动加1。

#1：创建一个系表

```
create table t_sdept(
id int primary key auto_increment,
sdept_name varchar(10),
s_num int);
```

#2.统计每个系多少人，然后插入到系表中

```
insert into t_sdept(sdept_name,s_num)
select sdept,COUNT(sno) from s group by sdept;
```

#3.创建触发器

```
create trigger tri_s
after insert on s
for each row
update t_sdept set s_num=s_num+1 where sdept_name=new.sdept;
```

#4.测试

```
select * from t_sdept;
insert into s
values(10,'aaaa','女',19,'IS');
```

删除触发器

- 删除语法

DROP TRIGGER [*schema_name*.]*trigger_name*

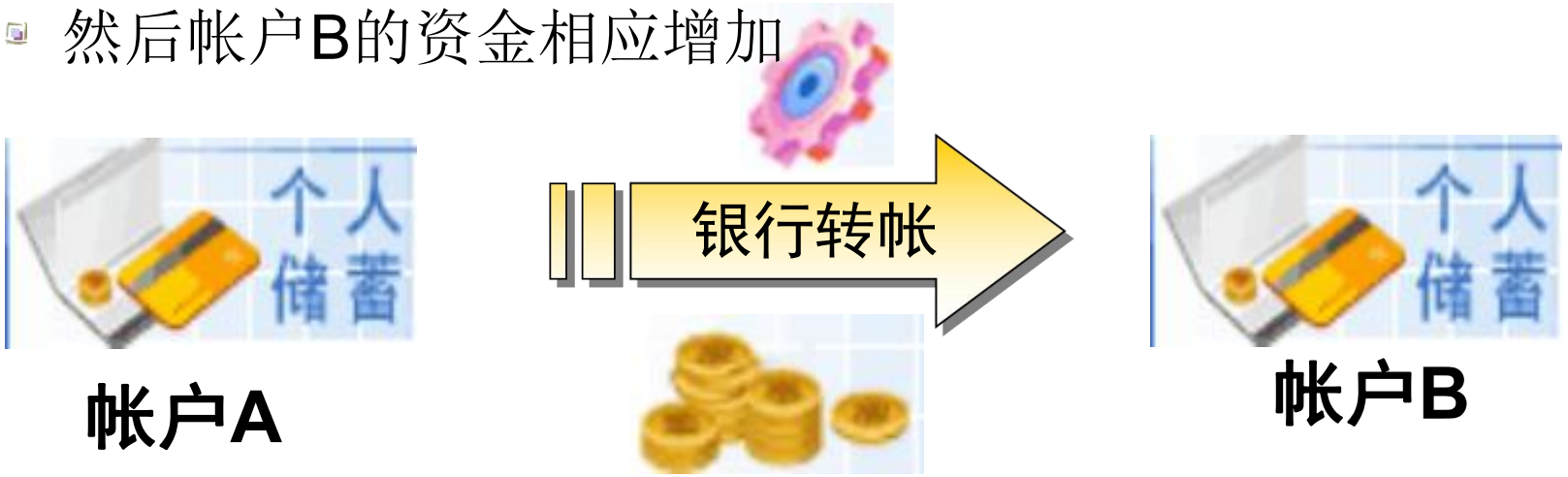
事务

所谓的**事务**就是针对数据库的一组操作，它可以由一条或多条SQL语句组成，**同一个事务的操作具备同步的特点**，即事务中的语句要么都执行，要么都不执行。

- **例如，银行转帐问题：**

  假定资金从帐户A转到帐户B，至少需要两步：

  - 帐户A的资金减少
  - 然后帐户B的资金相应增加



帐户A　　**帐户B**

- 假定张三的帐户直接转帐1000元到李四的帐户

创建帐户表，存放用户的帐户信息

```
--创建账户表
create table account (
 id int primary key auto_increment,
 name varchar(20),
 money float);
-- 表中插入值
insert into account(name,money)
values('张三',1000),('李四',1);
select * from account;
```

#创建一个触发器(触发器的作用是使得每个账号的钱不少于1块钱)

添加约束：根据银行规定，帐户余额不能少于1元，否则视为销户

```
# delimiter$$表示将语句的结束符;改成$$
Delimiter $$
create  trigger tri_account_money before update
on account
for each row
begin
  if new.money<1 then
      set new.money= old.money;
  end if;
end$$
```

为什么需要事务

- 模拟实现转帐：

  从张三的帐户转帐**1000**元到李四的帐户

```
/*--转帐测试：张三转账1000元给李四--*/
--我们可能会这样这样编写语句
--张三的帐户少1000元，李四的帐户多1000元
update account set money=money-1000
where name='张三';
 update account set money=money+1000
where name='李四';
--再次查看转帐后的结果。
SELECT * FROM account
```

请问：
执行转帐语句后，张三、李四的帐户余额为多少？

转账前：

id	name	money
1	张三	1000
2	李四	1

转账后：

id	name	money
1	张三	1000
2	李四	1001

- 张三的帐户没有减少
- 但李四的帐户却多了1000元
- 1000＋1001＝2001元
  总额多出了1000元！

为什么需要事务

- 错误原因分析：

```
#创建一个触发器(触发器的作用是使得每个账号的钱不少于1块钱)
##  delimiter$$表示将语句的结束符;改成$$
delimiter$$
create  trigger tri_account_money before update
on account
for each row
begin
  if new.money<1 then
    set new.money= old.money;
  end if;
end$$
```

检查UPDATE语句：
余额>=1元

```
--张三的帐户减少1000元，李四的帐户增加1000元
updat
set m
where
updat
set money=money+1000
where name='李四'
```

发器进行

为1001元

如何解决呢？使用事务

什么是事务

- 事务(TRANSACTION)是作为单个逻辑工作单元执行的一系列操作
- 这些操作作为一个整体一起向系统提交，要么都执行、要么都不执行
- 事务是一个不可分割的工作逻辑单元

转帐过程就是一个事务。

它需要两条UPDATE语句来完成，这两条语句是一个整体，如果其中任一条出现错误，则整个转帐业务也应取消，两个帐户中的余额应恢复到原来的数据，从而确保转帐前和转帐后的余额不变，即都是1001元。

事务的特性

事务必须具备以下四个属性，简称ACID 属性：

- 原子性（Atomicity）：事务是一个完整的操作。事务的各步操作是不可分的（原子的）；要么都执行，要么都不执行
- 一致性（Consistency）：当事务完成时，数据必须处于一致状态
- 隔离性（Isolation）：对数据进行修改的所有并发事务是彼此隔离的，这表明事务必须是独立的，它不应以任何方式依赖于或影响其他事务
- 永久性（Durability）：事务完成后，它对数据库的修改被永久保持，事务日志能够保持事务的永久性

如何创建事务

- SQL使用下列语句来管理事务：
  - 开始事务：START TRANSACTION 或者 begin work
  - 提交事务：COMMIT
  - 回滚（撤销）事务：ROLLBACK

一旦事务提交或回滚，则事务结束。

- 自动关闭和开启事务

- 默认情况下，每条单独的SQL语句视为一个事务

  关闭默认提交状态后，可手动开启、关闭事务

  语法：关闭/开启自动提交状态 SET autocommit = 0|1;

  值为0：关闭自动提交

  值为1：开启自动提交

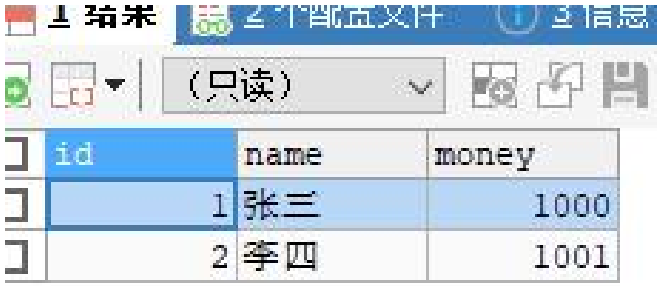
如何创建事务

- 使用事务解决银行转帐问题

```
-- 模拟转账：张三转账1000元给李四
 update account set money=money-1000 where name='张三';
 update account set money=money+1000 where name='李四';

-- 转账后账户余额：
select * from account;
rollback;
```

此时rollback没有起到回滚的作用，因为默认状态下MySQL自动提交事务；



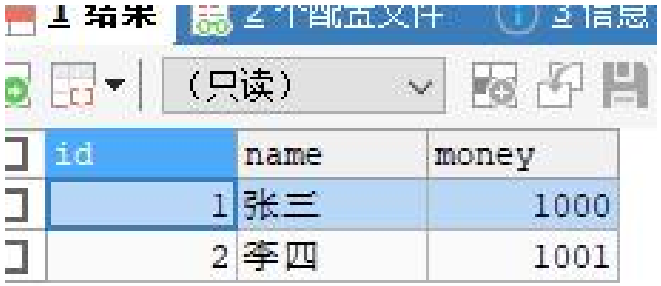
如何创建事务

- 使用事务解决银行转帐问题

```
-- 模拟转账：张三转账1000元给李四
 update account set money=money-1000 where name='张三';
 update account set money=money+1000 where name='李四';

-- 转账后账户余额：
select * from account;
rollback;
```

此时rollback没有起到回滚的作用，因为默认状态下MySQL自动提交事务；



如何创建事务

```
start transaction;  #手动开启事务
update account set money=money-1000 where name='张三';
update account set money=money+1000 where name='李四';
select * from account;
rollback;   #手动结束事务
select * from account;
```

id	name	money
1	张三	1000
2	李四	1001

id	name	money
1	张三	1000
2	李四	1

如何创建事务

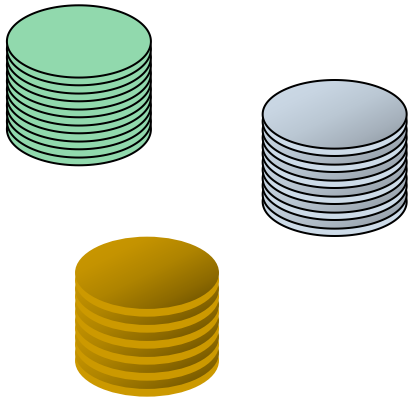
- 演示：转帐1000，转帐失败的情况

```
#转账演示（转账失败rollback）
begin;
select * from account;
update account set money=money-1000 where name='张三';
update account set money=money+1000 where name='李四';
select * from account;
rollback;   #手动结束事务
select * from account;
```

如何创建事务

- 演示：转帐100，转帐成功的情况

```
#转账演示（commit）张三给李四转100元
start transaction;
select * from account;
update account set money=money-100 where name='张三';
update account set money=money+100 where name='李四';
select * from account;
commit;
select * from account;
rollback; #一旦commit了，那么rollback不起作用，数据永久保存。
select * from account;
```



数据库系统

第九章 权限管理

北京工业大学耿丹学院
计算机科学与技术专业

- 在对 MySQL 的日常管理和实际操作中，为了避免用户恶意冒名使用 root 账号控制数据库，通常需要创建一系列具备适当权限的账号，应该尽可能地不用或少用 root 账号登录系统，以此来确保数据的安全访问。

本章内容

- MYSQL权限管理概述
- MYSQL中用户管理
- 用户权限的基本操作

一、MYSQL权限管理概述

- MYSQL中提供了一套非常实用的权限管理系统
- 通过该权限系统
  - 可以管理和控制某个用户使用其所提供的主机名、用户名和密码能否连接到指定的数据库服务器
  - 还能控制连接到数据库服务器的用户能否打开所需数据库以及能否对数据进行读取（select）、添加（insert）、修改（update）和删除（delete）等操作。

一、MYSQL权限管理概述

- MYSQL提供的权限以字段形式存储于系统数据库MySQL的user表中

```
-- 查看数据库中用户信息
select * from mysql.user;
```

Host	User	Select_priv	Insert_priv	Update_priv	Delete_priv	Create_priv	Drop_priv	Reload_priv	Shutdown_priv	Process_priv	File_priv
localhost	root	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
localhost	mysql.session	N	N	N	N	N	N	N	N	N	N
localhost	mysql.sys	N	N	N	N	N	N	N	N	N	N
localhost	test	N	N	N	N	N	N	N	N	N	N

```
-- 查看当前登录的用户select USER();
 select user();
```

二、用户管理

1.创建用户

CREATE USER <用户名> **[IDENTIFIED BY** 口令]

说明：

如果想远程登录，将"localhost"改为"%"，表示在任何一台电脑上都可以登录。也可以指定某台机器（例如192.168.1.10），或某个网段（例如192.168.1.%）可以远程登录。

1) <用户名>

指定创建用户账号，格式为'user_name'@'host_name'。

这里user_name是用户名，host_name为主机名，即用户连接MySQL 时所在主机的名字(即允许客户端连接的IP地址，如localhost,127.0.0.1)。若在创建的过程中，只给出了账户的用户名，而没指定主机名，则主机名默认为 “%”[代表所有ip都可以访问

2) identified by会将纯文本密码加密作为散列值存储

实例：创建用户

- 例 1 ：创建用户test，密码'1234'

```
create user 'test' @'localhost' identified by '1234';

create user 'test1' identified by '1234';

create user 'test2';
```

- -- 查看数据库中用户信息

```
select * from mysql.user;
```

Host	User	Select_priv	Insert_priv	Upd
localhost	root	Y	Y	Y
localhost	mysql.session	N	N	N
localhost	mysql.sys	N	N	N
localhost	test	N	N	N
%	test1	N	N	N
%	test2	N	N	N

实例：创建用户

- 例2：可以同时创建多个用户

```
create user 'guest1'@'localhost' identified by '123',
'guest2' identified by '123',
'guest3';
```

- -- 查看数据库中用户信息

```
select * from mysql.user;
```

Host	User	Select_priv	Insert_priv	Update_priv	Delete_priv	Create_priv	Drop_priv	Reload_priv	Shutdown_priv
%	guest2	N	N	N	N	N	N	N	N
%	guest3	N	N	N	N	N	N	N	N
localhost	guest1	N	N	N	N	N	N	N	N

二、用户管理

2.修改用户名

Rename user 旧用户名 to 新用户名[,旧用户名 to 新用户名]……

- **例2：修改刚才test 用户名为guest**

```
rename user test@localhost to guest@localhost
```

二、用户管理

3.修改密码

set password for ‘user’@‘host_name’ =‘新密码';

- 例3：修改用户密码为‘111'

```
create user 'test' @'localhost' identified by '1234';

create user 'test1' identified by '1234';

create user 'test2';
```

```
set password for 'test'@'localhost' ='111';
set password for 'test' ='111';

set password for 'test1' ='111';
set password for 'test1'@'localhost' ='111';
set password for 'test1'@'%' ='111';
```

二、用户管理

4. 删除用户

drop user <用户名>

说明：

用户名格式为

•创建的时候如果指定了主机名，则删除时的格式为

user_name@host_name；

•如果没有指定主机名，删除的时候也不需要给出，直接是用户名即可；

•一次可以删除多个用户。

实例：删除例 1 创建的三个用户

```
create user 'test' @'localhost' identified by '1234';

create user 'test1' identified by '1234';

create user 'test2';
```

```
drop user test    #错误，必须指出主机名
drop user test@localhost; #正确
drop user test1,test2; #创建的时候没有指定主机名，因此删除的时候也不需要给出
-- 或者一次可以删除多个用户
drop user test@localhost,test1,test2;
```

- -- 查看数据库中用户信息

```
select * from mysql.user;
```

三、权限管理

- MySQL中的权限信息被存储在mysql数据库的user、db、host、tables_priv、column_priv和procs_priv表中，当MySQL启动时会自动加载这些权限信息，并将这些权限信息读取到内存中。接下来通过下表列举一下MySQL的相关权限以及在user表中对应的列和权限范围。

MySQL 的权限信息

user 表的权限列	权限名称	权限范围
Create_priv	CREATE	数据库、表、索引
Drop_priv	DROP	数据库、表、视图
Grant_priv	GRANT OPTION	数据库、表、存储过程
References_priv	REFERENCES	数据库、表
Event_priv	EVENT	数据库
Alter_priv	ALTER	数据库
Delete_priv	DELETE	表
Insert_priv	INSERT	表
Index_priv	INDEX	表
Select_priv	SELECT	表、列
Update_priv	UPDATE	表、列
Create_temp_table_priv	CREATE TEMPORARY TABLES	表
Look_tables_priv	LOCK TABLES	表
Trigger_priv	TRIGGER	表
Create_view_priv	CREATE VIEW	视图
Show_view_priv	SHOW VIEW	视图
Alter_routine_priv	ALTER ROUTINE	存储过程、函数
Create_routine_priv	CREATE ROUTINE	存储过程、函数
Execute_priv	EXECUTE	存储过程、函数
File_priv	FILE	范围服务器上的文件
Create_tablespace_priv	CREATE TABLESPACE	服务器管理
Create_user_priv	CREATE USER	服务器管理
Process_priv	PROCESS	存储过程和函数

三、权限管理

- 针对表中部分权限进行分析，具体如下：
- **CREATE和DROP权限**，可以创建数据库、表、索引，或者删除已有的数据库、表、索引。
- **INSERT、DELETE、UPDATE 、SELECT权限**，可以对数据库中的表进行增删改查操作。
- **INDEX权限**，可以创建或删除索引，适用于所有的表。
- **ALTER权限**，可以用于修改表的结构或重命名表。
- **GRANT权限**，允许为其它用户授权，可用于数据库和表。
- **FILE权限**，被赋予该权限的用户能读写**MySQL**服务器上的任何文件。

三、权限管理

1.用户权限的授予

```
GRANT priv_type  [（column__list）]
ON <database.table|database.*|*.*|*>
TO user_or_role [,  user_or_role ] . . .
[WITH grant option];
```

- 功能：将对指定操作对象的指定操作权限授予指定的用户。

三、权限管理

priv_type ： 表示权限类型；

Columns：参数表示权限作用于某一列，该参数可以省略不写，此时权限作用于整个表；

TO user_or_role:表示授予某个用户或某个角色，可以同时给多个用户或多个角色授予权限。

-- Mysql8.0之前支持授权的同时创建用户

Mysql8.0之后，必须先创建用户，然后再授权，需要分开写。

WITH GRANT OPTION：将自己的权限授予其他用户。

实例：授权用户权限

练习，创建一个用户u1,密码是111

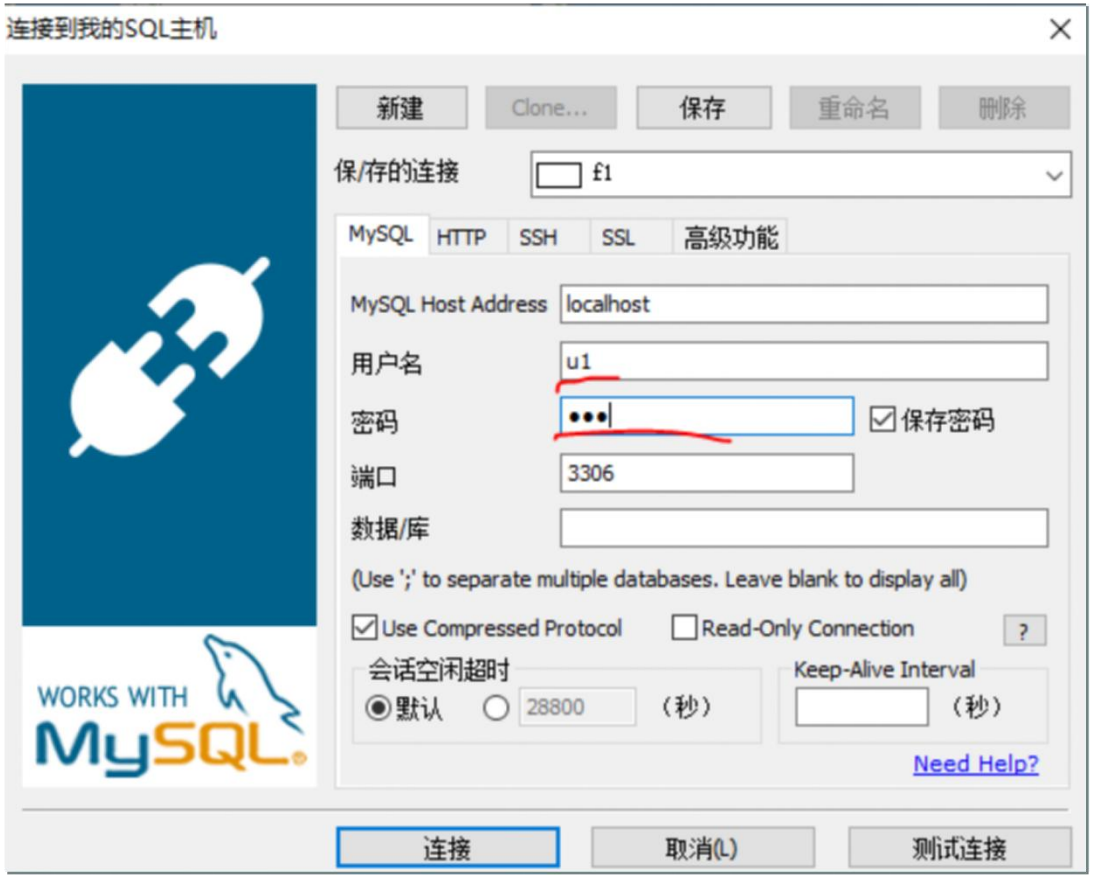
```
-- 法一：
create user u1 identified by '111';
-- 法二：
create user u2@localhost identified by '111';
```

- -- 查看数据库中用户信息

```
select * from mysql.user;
```

实例：授权用户权限

-- 1.通过新用户u1新建一个连接,登录数据库服务器



实例：授权用户权限

-- 2.通过用户u1登录的服务器中查询stu_cs库中学生表的基本信息

```
select * from stu_cs.s;
```

```
1 queries executed, 0 success, 1 errors, 0 warnings

查询: select * from stu_sc.s LIMIT 0, 1000

错误代码: 1142
SELECT command denied to user 'u1'@'localhost' for table 's'

执行耗时   : 0 sec
传送时间   : 0 sec
总耗时     : 0 sec
```

实例：授权用户权限

-- 3.给u1授予查询stu_cs库中学生表的基本信息的权限

```
-- root用户是超级管理员，可授予u1用户权限
-- 授予用户对表s查询权限
 grant select on stu_cs.s to u1
```

```
-- 查询权限授予
show grants for u1;
```

-- 验证

实例：授权用户权限

-- 4.授权用户u1对stu_cs数据库中表c中课程号和课程名的查询权限

```
grant select(cno,cname) on stu_cs.c to u1;
```

-- 5.授权用户u1对stu_cs数据库所有表的查询、插入、更新、删除权限

```
grant select,insert,update,delete on stu_cs.* to u1;
```

授予权限实例

- 6：授权某用户g具有对所有数据库中表的查询权限

- 7：授权用户aaa对所有数据库的操作权限，即超级用户权限

```
grant all privileges on *.* to aaa@'localhost' identified by "aaa";
```

权限的转移与限制实例

- 8：创建一个新的用户guest,密码是‘111’对数据库stu_mis中的表具有查询权限，并且允许将自身的权限授予其他用户。

```
create user guest identified by '111';
grant select on stu_mis.* to guest
with grant option;
```

- -- 9.创建一个新用户test1,主机名localhost,密码111

```
create user test1@localhost identified by '111';
```

- -- 10.用户guest授权test1具有查询stu_mis数据库的权限

```
grant select on stu_mis.* to test1@localhost;
```

查看权限

- SHOW GRANTS的语法格式如下：
  - SHOW GRANTS FOR 'username'@'hostname';

三、权限管理

2.用户权限的收回

Revoke priv_type　［（column__list）］

ON <database.table|database.*|*.*|*>

from user［，user］．．

- 功能：从指定用户那里收回对指定对象的指定权限

实例：收回用户权限

-- 1.查看用户u1的权限

```
-- 查询权限授予
show grants for u1;
```

Grants for u1@%
GRANT SELECT, CREATE ON *.* TO 'u1'@'%'
GRANT SELECT ON `mysql`.* TO 'u1'@'%'
GRANT SELECT ON `stu`.* TO 'u1'@'%'
GRANT SELECT ON `stu_mis`.* TO 'u1'@'%'
GRANT SELECT ON `stu_mis`.`s` TO 'u1'@'%'

-- 2.撤销u1对stu.*的查询权限

```
revoke select on stu.* from u1;
```

Grants for u1@%
GRANT SELECT, CREATE ON *.* TO 'u1'@'%'
GRANT SELECT ON `mysql`.* TO 'u1'@'%'
GRANT SELECT ON `stu_mis`.* TO 'u1'@'%'
GRANT SELECT ON `stu_mis`.`s` TO 'u1'@'%'

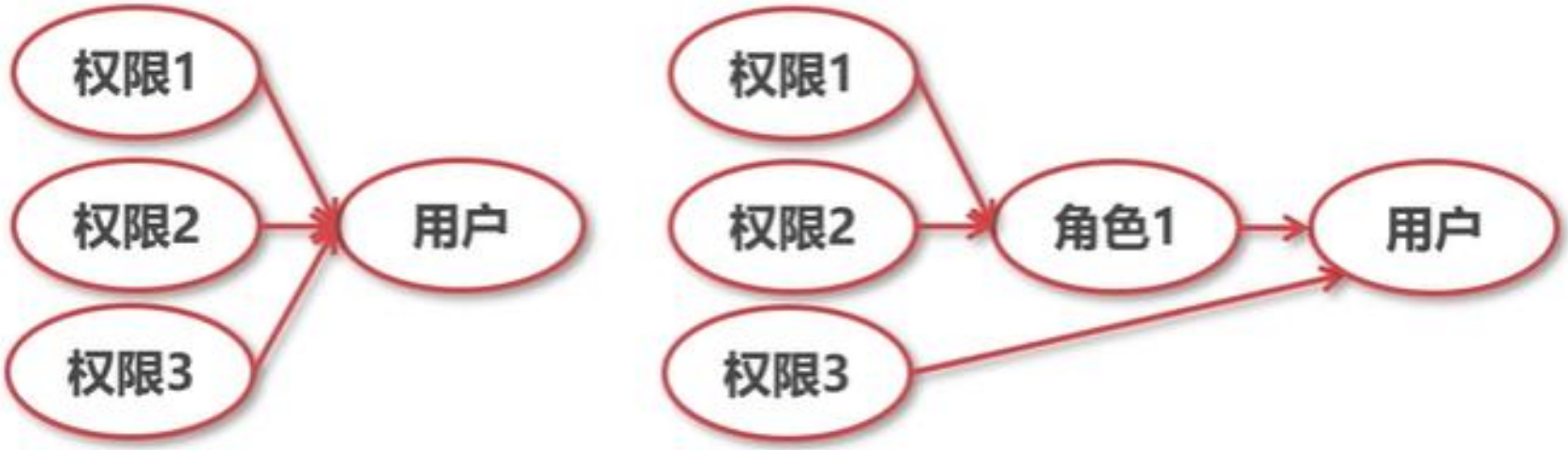
-- 验证

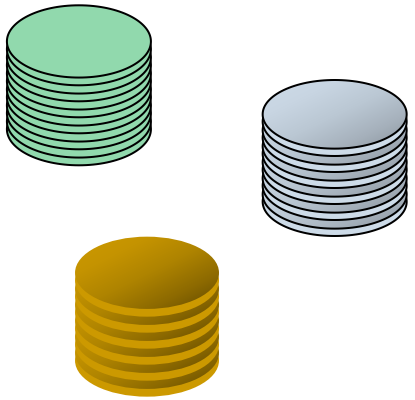
角色

- MySQL 8.0提供了角色管理的新功能，角色是一组权限的集合。
- 通过角色可以给多个账户授权，即将该角色的权限授权给其它账户。
- 权限的修改直接通过修改角色来实现，不需要每个账户一个一个的grant权限，方便运维和管理。
- role可以创建、删除、修改并作用到他管理的账户上。

角色

- MySQL 8.0提供了角色管理的新功能，角色是一组权限的集合。





数据库系统

第十章 数据库的备份和还原

北京工业大学耿丹学院
计算机科学与技术专业

本章学习目标：

- 掌握数据的备份操作
- 掌握数据的还原操作

数据的备份和还原

- 在操作数据库时难免会出现一些意外情况，导致数据丢失。
- 例如管理员操作失误、病毒入侵等不确定因素。
- 为了确保数据的安全，可以对数据进行定期备份。
- 这样当遇到意外情况时可以将数据还原，从而最大限度地减少损失。

1.以命令行的方式备份

- MySQL提供了mysqldump命令来实现数据的备份

2.以图形化的方式备份

以命令行的方式备份

Mysqldump备份整个数据库

Mysqldump备份整个数据库中各表的结构

Mysqldump备份多个数据库

Mysqldump全备所有库和所有表实例

mysqldump 命令执行时，可以将数据库中的数据备份成一个文本文件。数据表的结构和数据将存储在生成的文本文件中。

MySQLdump进行备份

- 运行mysqldump不是在mysql客户端，而是在cmd命令行运行即可下，它位于mysql的bin下，
- 如：C:/Program Files/MySQL/MySQL Server 8.0/bin>

```
C:\Users\User>cd C:\Program Files\MySQL\MySQL Server 8.0\bin
```

```
C:\Program Files\MySQL\MySQL Server 8.0\bin>mysqldump -help
Usage: mysqldump [OPTIONS] database [tables]
OR     mysqldump [OPTIONS] --databases [OPTIONS] DB1 [DB2 DB3...]
OR     mysqldump [OPTIONS] --all-databases [OPTIONS]
For more options, use mysqldump --help

C:\Program Files\MySQL\MySQL Server 8.0\bin>
```

Mysqldump的常用语法

说明：

mysqldump 的几种常用方法：

（1）导出整个数据库(包括数据库中的数据）

```
mysqldump -u username -p dbname > dbname.sql
```

（2）导出数据库结构（不含数据）

```
mysqldump -u username -p -d dbname > dbname.sql
```

（3）导出数据库中的某张数据表（包含数据）

```
mysqldump -u username -p dbname tablename > tablename.sql
```

（4）导出数据库中的某张数据表的表结构（不含数据）

```
mysqldump -u username -p -d dbname tablename > tablename.sql
```

Mysqldump备份整个数据库

- 语法格式：

mysqldump -u username -p dbname [tbname ...]> filename.sql

参数说明：

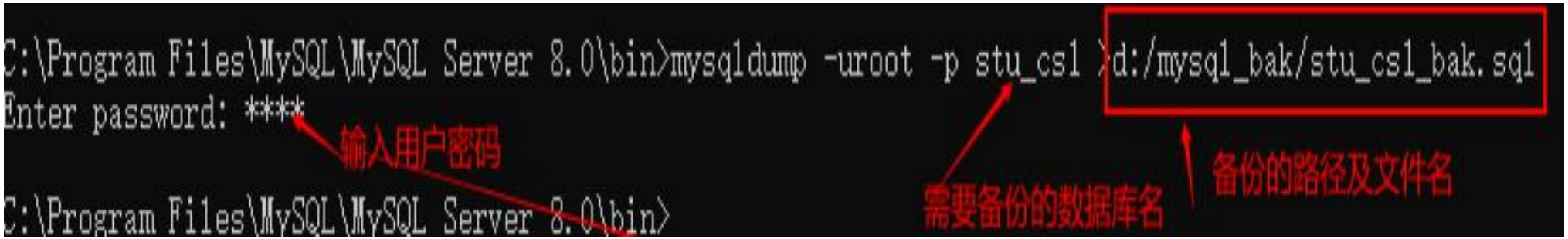
•username：表示用户名称；

•dbname：表示需要备份的数据库名称；

•tbname：表示数据库中需要备份的数据表，可以指定多个数据表。省略该参数时，会备份整个数据库；

•右箭头“>”：用来告诉 mysqldump 将备份数据表的定义和数据写入备份文件；

•filename.sql：表示备份文件的名称，文件名前面可以加绝对路径。通常将数据库备份成一个后缀名为.sql的文件。



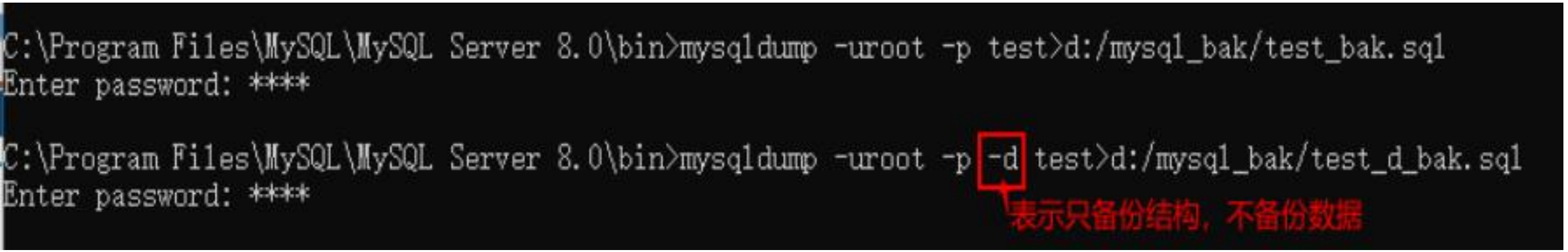
- **查看（通过sqlyog打开文件——stu_cs1_bak.sql）**

备份整个数据库的结构（不含数据）

mysqldump -u username -p –d bname [tbname ...]> filename.sql

参数说明：

•增加参数 –d：表示只备份数据库中各表的结构，不包含数据



- **查看**
- **（通过sqlyog打开test_d_bak.sql）**

备份数据库的某个表（包含数据）

- mysqldump -u username -p dbname tbname1 tbname2 ...]> filename.sql

参数说明：

•数据库后面直接跟表名，如果是多个表，用空格分隔

```
C:\Program Files\MySQL\MySQL Server 8.0\bin>mysqldump -uroot -p test s>d:/mysql_bak/test_s_bak.sql
Enter password: ****
```

- **查看（通过sqlyog打开test_s_bak.sql）**

思考：备份某数据库指定的多个表?

```
C:\Program Files\MySQL\MySQL Server 8.0\bin>mysqldump -uroot -p test s c>d:/mysql_bak/test_s_c_bak.sql
Enter password: ****
```

多个表之间用空格表示

备份数据库的某个表结构（不包含数据）

- mysqldump -u username -p -d dbname tbname1 tbname2 ...]> filename.sql

- 例如备份学生选课系统数据库中s表的结构

```
C:\Program Files\MySQL\MySQL Server 8.0\bin>mysqldump -uroot -p -d stu_cs1 s>d:/mysql_bak/stu_cs1_s_bak.sql
Enter password: ****
```

- **查看（通过sqlyog打开该备份文件）**

练习：备份学生选课数据库中s,c,sc表的结构?

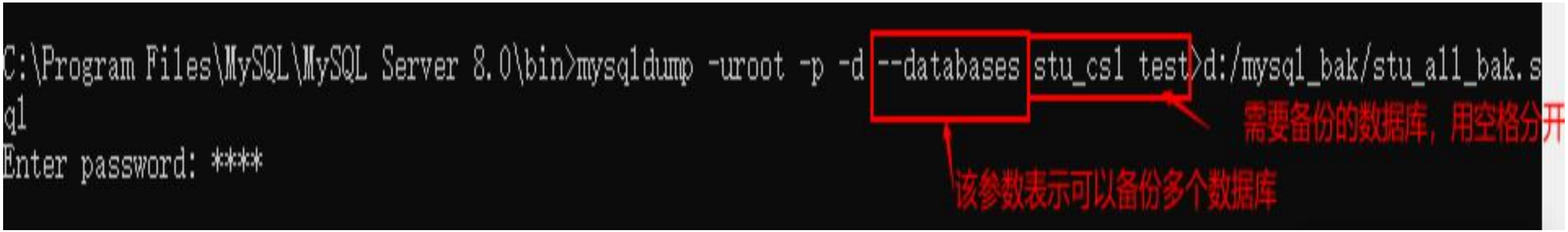
Mysqldump备份多个数据库

使用 **mysqldump** 命令备份多个数据库，需要使用 **--databases** 参数。

备份多个数据库的语法格式如下：

```
mysqldump -u username -p --databases dbname1 dbname2 ... > filename.sql
```

加上“--databases”参数后，必须指定至少一个数据库名称，多个数据库名称之间用空格隔开。



Mysqldump备份所有库和所有表实例

mysqldump 命令备份所有数据库的语法格式如下：

mysqldump -u username -P --all-databases>filename.sql

使用“--all-databases”参数时，不需要指定数据库名称。

```
C:\Program Files\MySQL\MySQL Server 8.0\bin>mysqldump -uroot -p --all-databases>d:/mysql_bak/all_bak.sql
Enter password: ****
```

2.以图形化的方式备份

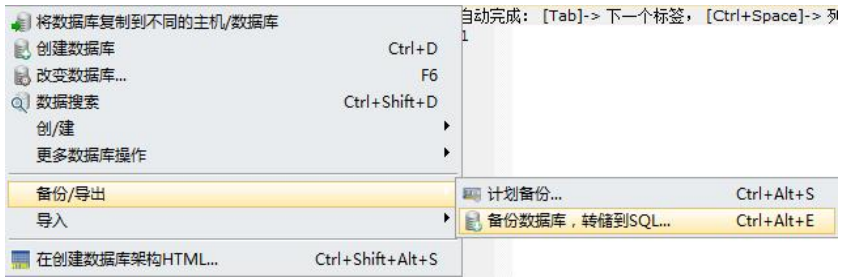
- MySQL客户端工具—SQLyog来做数据库备份，也就是以图形化的方式备份，这样使用起来更加便捷。

  - 首先进入SQLyog主界面。

2.以图形化的方式备份

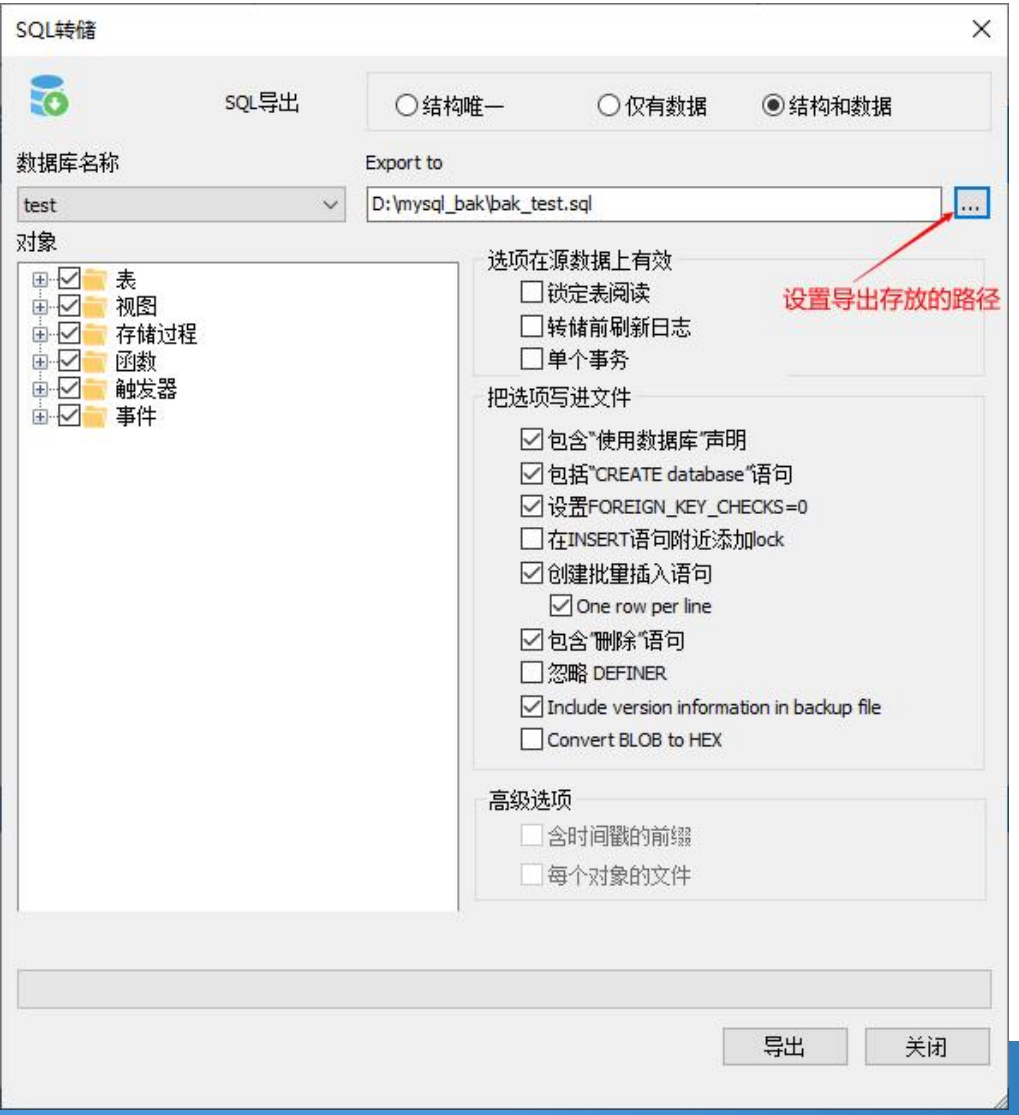
- 左侧导航栏中是MySQL中所有的库，此处需要备份的是test库，右击test，弹出下图所示的右键菜单。

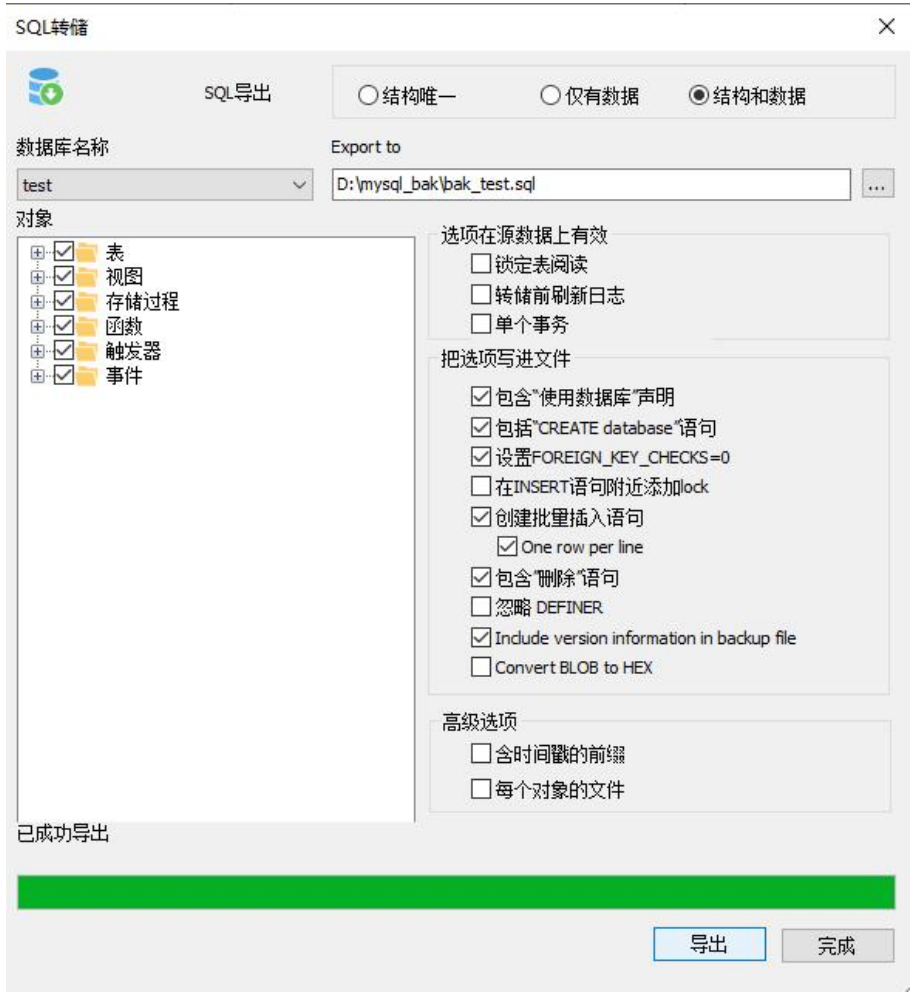
- 选择“备份/导出”，然后单击“备份数据库，转储到SQL...”命令。



2.以图形化的方式备份

- 此时出现SQL转储弹窗。设置导出文件的路径。





数据的还原

- 当数据丢失或意外损坏时，可以通过恢复已经备份的数据来尽量减少数据的丢失和破坏造成的损失。

1.以命令行的方式还原

- 使用mysql命令来实现数据的还原

2.以图形化的方式还原

以命令行的方式还原

- MySQL提供了mysql命令来实现数据的还原，具体语法格式如下。

```
mysql -u username -p dbname<filename.sql
```

- -u后的参数username表示用户名，
- -p后的参数password表示登录密码，
- dbname表示需要还原的数据库名称，
- filename.sql代表备份文件的名称,可以添加该文件的绝对路径。
- 在使用mysql命令还原数据库时，同样直接在DOS命令行窗口中执行即可，不需要登录到MySQL数据库。

数据还原案例

- 为了演示还原，需要先删除用于test库中表s, c。

```
drop table s,c;
```

- test库中没有任何数据表。下面进行数据还原，打开DOS命令行窗口，输入以下命令。

```
mysql -uroot -proot test<d:/mysql_bak/bak_test.sql
```

  - -u后的参数root表示用户名，-p后的参数root表示登录密码
  - test表示需要还原的数据库名称
  - d:/mysql_bak/代表备份文件存放的路径
  - bak_test.sql代表备份文件的名称。

可以登录sqlyog进行验证，是否数据恢复成功

Source命令还原数据

除了使用mysql命令外，还可以使用source命令还原数据。

同样将test库中表S的数据删除，然后，然后录到MySQL数据库，执行下面一系列操作来验证。

```
C:\Users\User>mysql -u root -proot
```
登录Mysql

执行完上述操作后，查看s表中的数据，发现s表中的数据还原成功。

```
mysql> select * from s;
+-----+-------+---------------------+
| sno | sname | stime               |
+-----+-------+---------------------+
|   1 | a     | 2021-12-26 17:20:21 |
|   2 | b     | 2021-12-26 17:20:28 |
+-----+-------+---------------------+
2 rows in set (0.00 sec)
```

```
命令提示符 - mysql -u root -proot

mysql> use test
Database changed
mysql> show tables;
+----------------+
| Tables_in_test |
+----------------+
| c              |
| s              |
+----------------+
2 rows in set (0.00 sec)

mysql> select * from s;
+-----+-------+---------------------+
| sno | sname | stime               |
+-----+-------+---------------------+
|   1 | a     | 2021-12-26 17:20:21 |
|   2 | b     | 2021-12-26 17:20:28 |
+-----+-------+---------------------+
2 rows in set (0.00 sec)

mysql> delete from s;
Query OK, 0 rows affected (0.00 sec)

mysql> select * from s;
Empty set (0.00 sec)

mysql> source d:/backmysql/bak_test.sql;
Query OK, 0 rows affected (0.00 sec)
```

第一步：选择test数据库

第二步：查看test库中的表

第三步：查看表s中的数据

第四步：删除表s中的数据

第五步：查看表s中的数据是否删除成功

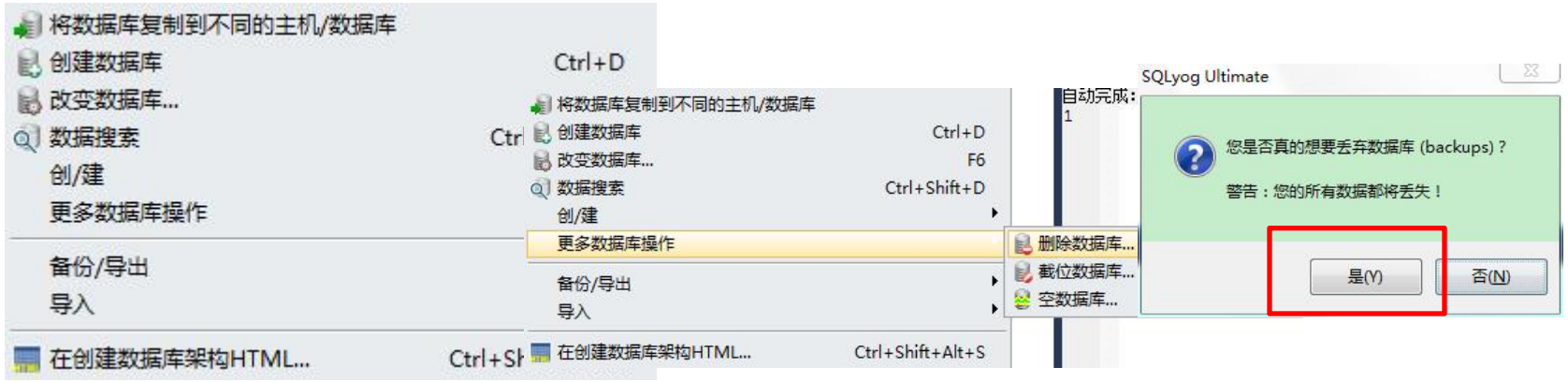
利用source命令进行数据恢复

使用source命令和mysql命令的区别是：source命令需要登录MySQL数据库后使用。

2. 图形化的方式还原

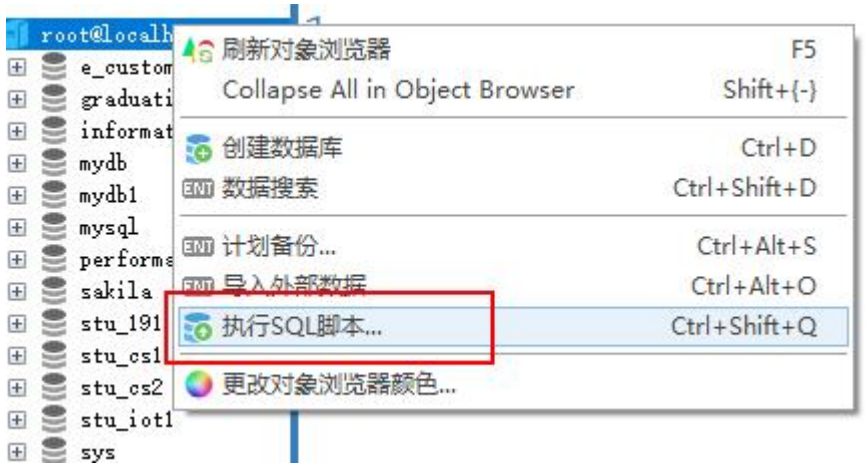
- 同样的，数据还原也可以通过图形化的方式实现，也就是使用SQLyog还原数据。

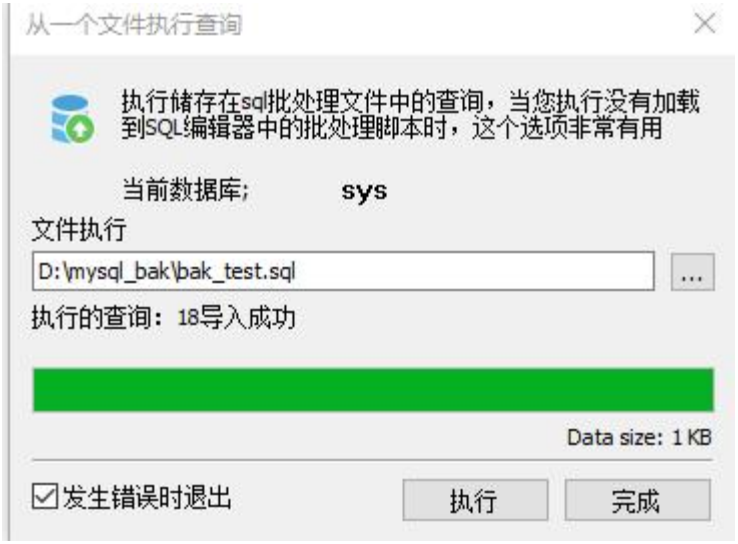
- 首先删除test库。
- 右击test库→更多数据库操作-→删除数据库-→是。



2.图形化的方式还原

- 右击“root@localhost”。





2.图形化的方式还原

- 此时按“F5”刷新数据库，然后可以看到test库及表已经还原，如图所示。

- 打开数据表s,可以查看表中数据