# 软件设计

# 数据库分析与设计

## 概述

一个数据库应用系统的设计包含俩方面的内容:

1. 结构特性的设计,即数据库模式或数据结构的设计
2. 行为特性的设计,即应用程序,事务处理的设计

采用UML进行系统的分析与设计,可以把系统的应用程序(行为)设计和数据库设计统一起来

数据模型(Data Model)是对现实世界的抽象,是建立数据库的基础,通常在一个数据模型中需要描述数据的组织结构,对数据的操作和数据的完整性约束

当前存在的数据模型大体可以分为俩类:

1. 基于记录的逻辑模型

传统的三种数据模型:层次模型,网状模型和关系模型

他们是以数据记录和数据项作为基本的数据结构

1. 基于对象的逻辑模型:

实体联系模型(ER模型),面向对象数据模型等

他们是更高抽象层次意义上的概念模型

### 传统的数据库设计

传统的关系数据库设计的过程分为四个阶段:需求分析,概念结构设计,逻辑结构设计,物理结构设计

需求分析阶段: 对数据的存储要求和处理要求进行描述,编制出数据库需求分析说明书

数据库的概念结构设计:

或成为企业的组织模式,它是独立于任何一种数据模型和任何具体的DBMS的信息结构模型,是现实世界的纯粹表示

需要借助于某种工具或方法,当前应用最广泛的是ER(实体-联系方法),使用ER图定义企业的信息组织模式

### 使用UML设计数据库

做法:

1. 以ER图为基础进行数据库设计,设计工作按照er模型的概念和方法进行,只是用uml表达
2. 用uml直接进行数据库的分析设计和表达

第二种做法可以把数据库设计与应用系统软件的设计统一进行

#### 综述：

UML的对象类图能够更好地用于对数据库建模,UML的对象类图不但对数据,而且能对行为建模.这些行为在物理数据库中被设计成触发器和存储过程,或者专用的Utility

#### 设计步骤

使用UML进行数据库设计的四个阶段:

##### 业务use case 模型设计

使用use case图,类图，顺序图，活动图等建立use case 模型

##### 逻辑数据模型设计

确定应用所需要的持久数据,对于RDBMS而言,需要设计出表达持久数据的实体类(Entity Class)及其联系,并把它们映射成为关系数据库表,视图等

##### 物理数据模型设计

使用类图 组件图 配置图等设计数据库的物理模型

##### 物理实现设计

把物理数据模型转换到实际的数据库环境

### UML用于数据库设计的规范

#### 表

表是RDBMS的基本建模结构

表又称为二维表,关系，它是具有相同结构的行的集合

表中的每一列称为关系的属性

任一行与列交叉点的元素称作分量,每一个数据必须作为一个分量值存储

每一个表有一个关系模式(Relational Scheme),它由一个表名以及它所有的列明构成,它是一个二维表的表头,规定该表的组成,列的特性,以及完整性约束等

表结构及相关的术语如下图:



##### 要点

在uml中表用类表示,带有构造型<<Table>>,类名即表名,类的属性描述表的列特性,包括列名,数据类型,以及有关的约束,在类的行为部分,根据需要可以给出对表的一些操作,也可以缺省.

如学生表和课程表的UML类图如下:



#### 关键字与索引

##### 概述:

###### 关键字(key)

关键字(key)用于存取表中的数据

主关键字(primary key)是一个属性或属性组,他的值唯一地标识表中的行

外来关键字(foreign key)是一个属性或属性组,她在本表中不是关键字,不能唯一地表示表中的行,但它在另一个表中是主关键字,能够唯一地表示该表中的行,外来关键字体现了俩个表的联系,实现表之间的参照的完整性.

主关键字和外来关键字的表示也可以分别使用构造型<<pk>>和<<fk>>.

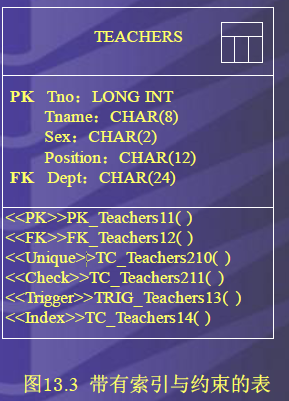
###### 索引

索引(Index)支持快速数据存取的物理结构

在表中可以指定某个属性作为索引项,或指定某几个属性组成多级索引项

在UML中索引的构造型为<<Index>>

如图:



###### 约束

约束(Constraint)是一种施加于数据库结构的规则,用于保证数据库的完整性.

约束描述数据库的动态行为,而表和列只表示数据库的静态结构方面

主关键字约束

主关键字约束定义一个表的主关键字,没个表中只允许有一个主关键字.

约束使用构造型<<PK>>

外来关键字约束

外来关键字约束是实现表之间联系的约束,通常定义在子表中.

外来关键字是这样的列,它在父表中是主关键字,而在子表中则不是关键字.

外来关键字使用<<Fk>>构造型

#### 联系

##### 概述:

表之间的任何依赖关系称为联系(Relationship)

一个联系是一个带有联系构造型的关联(Association)和一组主关键字,外来关键字的整合

每一个联系都存在于父表和子表之间,父表中必须定义有主关键字,而子表则有外来关键字列和外来关键字约束,用于访问父表.

##### 联系

联系又俩种:非确定联系和确定联系

###### 非确定联系(Non-Identify relationship)

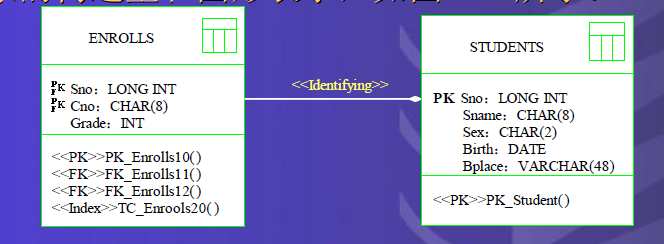
非确定联系是存在于俩个独立表之间的联系,在子表中的外来关键字为非主关键字.

###### 确定联系

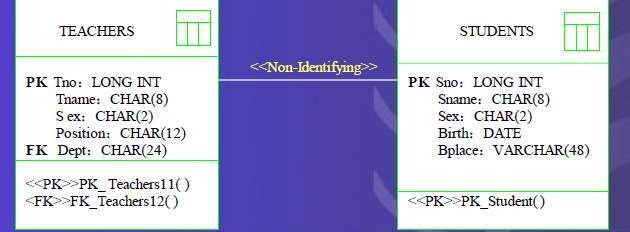
是存在于有依赖的俩个表之间的联系,若父表不存在,则子表不可能存在,子表的存在依赖于父表的存在,在父表中的所有的主关键字列成为子表的主关键字列和外来关键字列.

###### 构造型和图形表示

确定联系的图示



非确定联系的构造型和联系



#### 视图

##### 概述

视图是一个虚表,它代表具有相同结构的一组数据记录,只不过他的物理上的数据来源是某一个或多个表.

视图不是物理存在的,它不包含真正存储的数据,不占存储空间,但视图可以像一般的表那样操作(少有限制).

真正物理上存在的表称作实表或基本表.

在同一个基本表上可以创建多个视图,一个视图也可以在几张表上创建.此外,一个视图也可以从另一个视图创建

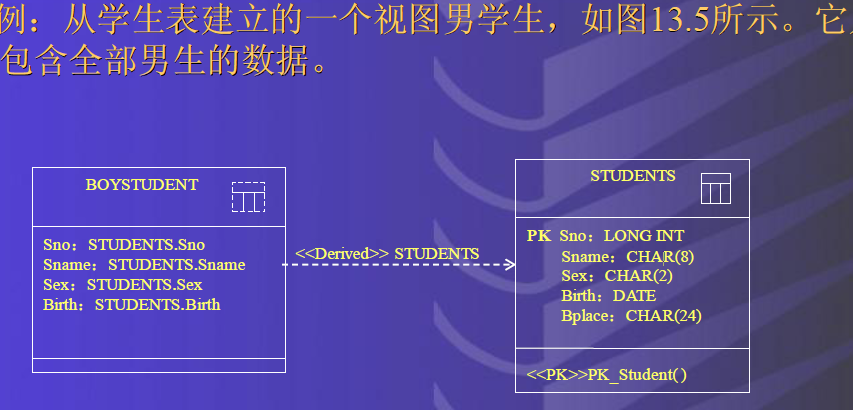
在uml中构造型为<<view>>

视图的属性必须与来源表中相应的属性一致。在视图的行为部分，根据需要可以给出一些操作，也可以缺省。视图与其存储来源数据的父表或父视图之间的关系用构造型<<Derived>>的依赖表示

##### 优点

视图像是数据库的一个窗口,他提高了数据库对于应用程序的独立性,有利于保持数据的一致性:简化了复杂查询,可以只在视图上查询,而不必对基本表查询,提供了一种保持数据库安全性的手段.

##### 图例:



### 数据库与数据库模式

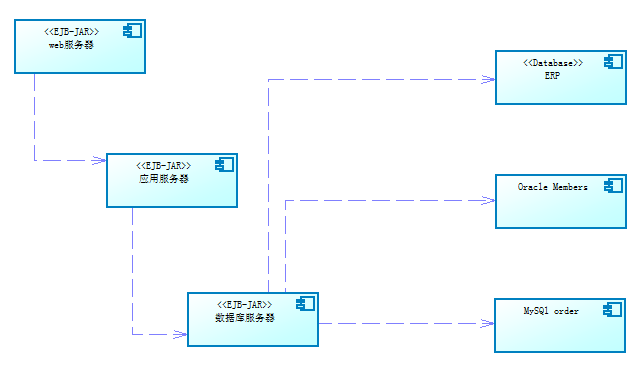
#### 数据库

##### 概述:

数据库(DB)是一种对用户透明的物理结构,用于存取数据,并控制对所存储的数据的存取.

在uml中,数据库用带有构造型<<Database>>的组件表示,也可以用一个柱形图标表示

##### 图例



#### 数据库模式

##### 概述:

数据库模式(Schema)是表的组织的基本单位,它存储了用于数据存储和检索的数据模型的完整的描述.

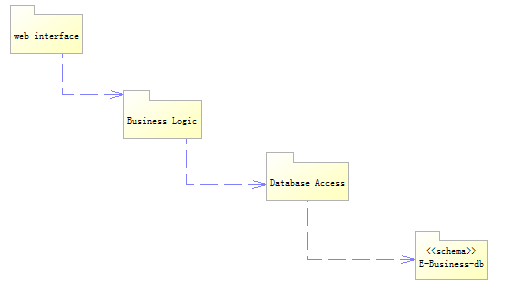
一个schema必须指派给一个数据库,而一个数据库可以有一个或多个schema

对于RDBMS来说,一个具体的relational Database 是一个关系的集合,而Relational Database schema 是关系模式的集合,它存储了对关系数据的描述,包括对表的静态结构和完整性约束的定义.

在uml中,schema用带有构造型<<schema>>的包表示.

##### 图例

一个schema的图例,构造型<<schema>>



### 表空间

#### 概述

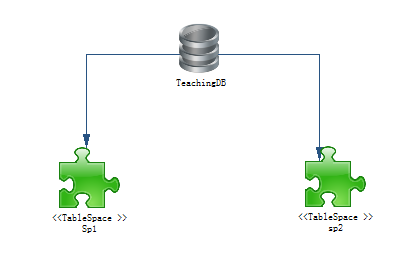
表空间(Tablespace)是指数据的存储区域,表空间代表数据库系统,它是数据库与节点之间的联系.

一个数据库可以分步在一个或多个表空间,这取决于数据库的数据量,数据访问要求和安全性要求.

表空间用带有构造型<<TableSpace>>的组件表示,表空间与数据库的联系用依赖表示

##### 图例

表空间,一个在俩个表空间上实现的数据库TeachingDB

表空间带有构造型<<TAblespace>>

### 节点

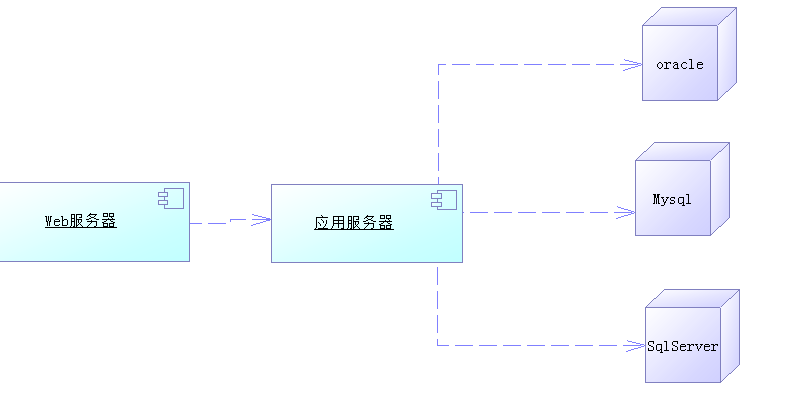
##### 概述

数据库驻留在节点(Node)上

节点在配置图中使用,通过节点和节点之间的联系,表达计算机系统的物理配置结构.

##### 图例

一个数据库系统的服务器端配置图



## 关系数据库逻辑模型设计

#### 逻辑数据模型设计

##### 概述

在逻辑数据模型设计阶段,主要任务是确定应用系统所需要长期保存的持久性数据,建立数据库的逻辑模型.

对于关系数据库,则需要根据系统中的实体类,创建关系数据库模式,包括表,视图,索引等数据结构元素的设计,以及相应的触发器,存储器等的设计.

要从众多的对象类中分离并确定具有确定持久性数据的实体类,把他们转换成相应的关系数据库表,并且根据实体类的属性的特征,以及类之间的联系，进行适当的映射与处理，从而建立起一个完整的关系数据库逻辑模型.

与数据的增删改等有关的约束控制,可以采用触发器实现,更一般的与数据存取有关的操作,可以采取存储过程实现.

在设计中应注意一下俩点:注意关系数据库的结构特点,注意数据库应用系统的性能.

#### 从类创建表

##### 概述

将类转换为表的最简单做法是一对一映射,即为每一个实体类创建一个表,类中的属性就是表中的列,类中的关键字就是表的关键字.

在进行属性的映射时,必须注意数据类型的转换,即要把对象类的属性在原来所采用的应用模型中数据类型,转换为数据库模型中的列的数据类型.

导出属性往往不必作为持久性数据存储在数据库中,当须要使用这些导出属性数据时,只要调用某个相关的程序进行计算推演即可,可以用存储过程或一个程序段实现.

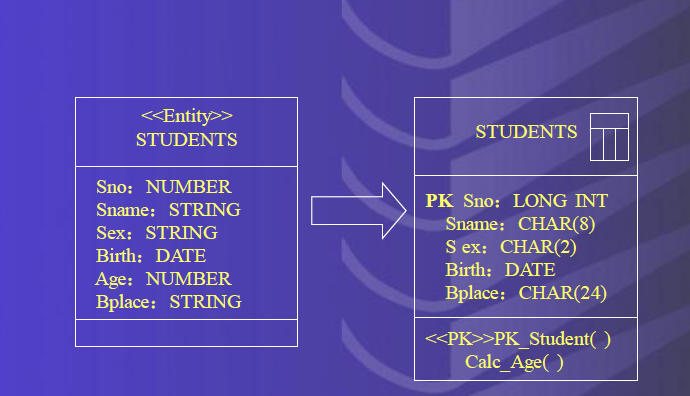
从类创建表的过程中,需要根据数据的完整性和安全性方面的要求,指定主关键字和外来关键字,增加必要的约束条件,如合法输入值校验等.

从类创建表的过程中,必须考虑类之间的联系,注意关系数据模型的特点.

必要时可以吧系统的逻辑设计模型中的一个类映射为多个表,或把多个类映射为一个表.

##### 图例

类向表映射的实例



#### 联系的映射

##### 概述

##### 关联的映射

对于持久数据的类模型,类之间允许有各种关联,参与关联的类的角色(Role)也可以是多种多样的.

关系数据模型只支持表之间的非确定联系和确定联系,以及多重性为”一对多”的情况.

把类之间的关联映射为表之间的非确定联系或确定联系时,对于非”一对多”的情况需要做一些适当的处理.

在关系数据模型中,无法直接实现表之间的确定联系或非确定联系,它们只能通过外来关键字或约束条件来实现.

##### 一对一映射

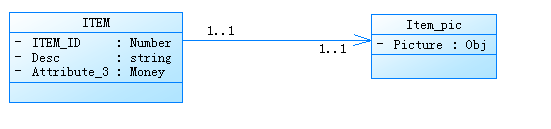
类之间的关联的多重性是”一对一”

在面向对象的设计中,”一对一”的关联代表俩个独立的对象类之间的联系.

在向关系数据模型的映射中,可以为每一个类创建一个表,其中每个表的主关键字也是它所关联的表的外来关键字,多重性”一对一”通过强制的约束实现.

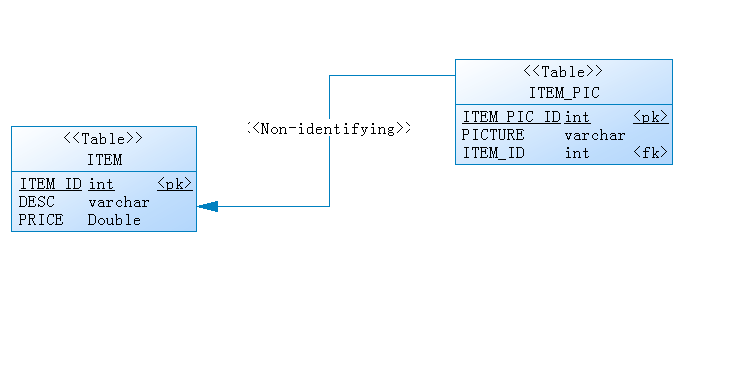
###### 图例

下图中,类ITEM与类ITEM\_pic存在”一对一”的关联,即一个物件(ITEM)必有一张物件图(ITEM\_PIC),而一张物件图比指定给一个确定的物件.



这个一对一的关联转化为关系数据模型

其中,表ITEM与表ITEM\_PIC之间的联系为非确定联系,它由在表ITEM\_pic中引入的外来关键字列ITEM\_ID建立,而ITEM\_ID是表ITEM中的关键字,同时在表ITEM\_PIC中生成外来关键字约束<<Fk>>FK\_ITEM.

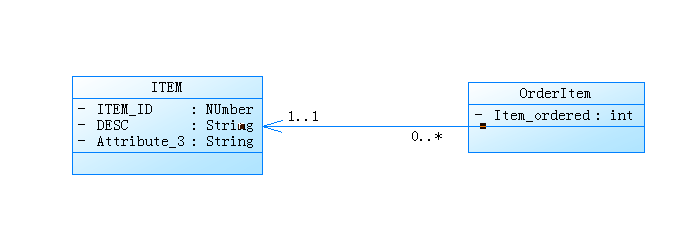


##### 一对多映射

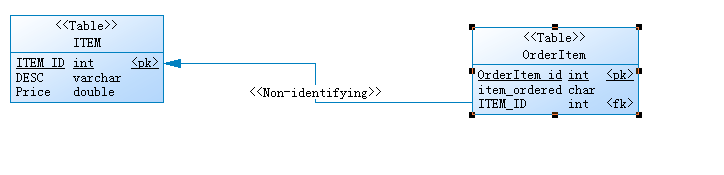
在向关系数据模型的映射中,可以为每一个类创建一个表,其中”一”方的表的主关键字也是它所关联的”多”方的表的外来关键字.

###### 图例

在下图中,类item与类Ordereditem存在”一对多”的关联,即一个已订购的物件ordereditem必与一个确定存在的物件item相关联,而一个物件可以被多次订购.



该一对多的关联转换为关系数据模型如下图



其中,表item与表ordereditem之间的联系为非确定联系,它由在表ordereditem中引入外来关键字字段ITEM\_id建立,同时在表ordereditem中生成外来关键字约束 <<Fk>>Fk\_ordereditem.

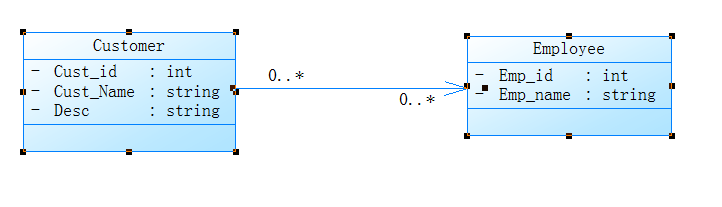
##### 多对多映射

再向关系数据模型的映射中,除了为每一个类创建一个表外,必须创建一个关联表(Associate Table),实现多对多的关联.

关联表的作用是把一个”多对多”的联系转换为俩个”一对多”的联系,该关联表的主关键字由它所关联的每个表的主关键字组成.

###### 图例

下图中类customer与类employee存在”多对多”的关联,一个商场的工作人员可以为多名顾客服务,而一个顾客可以得到多个商场的工作人员的服务



在下图中,表customer与表employee通过一个新增加的关联表cust\_emp实现”多对多”的关联,表”Customer”与表”cust\_emp”,表employee与表”cust\_emp”之间都是”一对多”的确定联系.

