--最后的胜利(1.3-1.9)

Header

期末考时搜集整合几篇博客+笔记而成，深度的知识区折叠了

删除了部分考试章节中没有的部分，同时偷懒了一点，毕竟目的是考试

由于相比于上一次计算机组成原理时间紧迫得很(只有5天复习)，因此部分内容只能等到日后复盘整理

1绪论

概念

数据（Data）

- 计算机用来描述事物的记录（文字．图形．图像．声音）
- 数据的形式本身并不能完全表达其内容，需要经过语义解释。数据与其语义是不可分的

数据库（Database，简称DB）

- 数据库是长期存储在计算机内有结构的大量的共享的数据集合。

数据库管理系统（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。
- 数据库在建立、运用和维护时由数据库管理系统统一管理、统一控制。
- 数据库系统（DBS）
  - 数据库系统是指在计算机系统中引入数据库后的系统构成，一般由数据库、数据库管理系统（及其开发工具）、应用系统、数据库管理员和用户构成。

数据冗余度

- 指同一数据重复存储时的重复程度。

数据的安全性（Security）

- 数据的安全性是指保护数据，防止不合法使用数据造成数据的泄密和破坏，使每个用户只能按规定，对某些数据以某些方式进行访问和处理。

数据的完整性（Integrity）

- 数据的完整性指数据的正确性、有效性和相容性。即将数据控制在有效的范围内，或要求数据之间满足一定的关系。

并发（Concurrency）控制

- 当多个用户的并发进程同时存取、修改数据库时，可能会发生相互干扰而得到错误的结果并使得数据库的完整性遭到破坏，因此必须对多用户的并发操作加以控制和协调。

数据库恢复（Recovery）

- 计算机系统的硬件故障、软件故障、操作员的失误以及故意的破坏也会影响数据库中数据的正确性，甚至造成数据库部分或全部数据的丢失。DBMS必须具有将数据库从错误状态恢复到某一已知的正确状态（亦称为完整状态或一致状态）的功能。

数据管理三阶段

- 人工管理阶段
- 文件系统阶段
- 数据库系统阶段

数据独立性特点– 不独立–›独立性差–›物理+逻辑独立性

数据独立性

略

- 物理独立性：是指用户的应用程序与数据库中数据的物理存储是相互独立的。
- 逻辑独立性：是指用户的应用程序与数据库的逻辑结构是相互独立的。
- 数据独立性是由数据库管理系统提供的二级映像功能来保证的。

> - 数据独立性是由DBMS的二级映像功能来保证的（外模式/模式映像，模式/内模式映像），这两层映像机制保证了数据库系统中数据的逻辑独立性和物理独立性。

数据独立性最高的数据模型是关系

DB三要素

- 数据（描述事物的符号记录，数据库里面存储的内容）
- 存储器（外存，一般是硬盘，数据库的载体）
- 数据库管理系统（DBMS，数据库的管理软件）

数据模型

数据模型是数据库系统的**核心**和**基础**

模型定义

- 是现实世界特征的模拟和抽象
- 数据抽象方法

  现实世界抽象到信息世界使用　　**分类**　　**概括**　　**聚集**

数据模型

- 也是一种模型，它是现实世界数据特征的抽象，表示实体以及实体间的联系
- 是一个用于描述数据、数据间关系、数据语义和数据约束的概念工具的集合

类型

- **概念模型**
  - 是现实到信息世界的第一层抽象,也是数据库设计人员和用户之间进行交流的语言
  - **这一类中最著名的是 实体 – 关系模型**

- 逻辑模型和物理模型

  逻辑模型

  - 层次模型
  - 网状模型
  - 关系模型
  - ……

  物理模型

  - 对数据最底层的抽象

概念模型

基本概念

- 实体（Entity）
  - 客观存在并可相互区别的事物称为实体
- 属性（Attribute）
  - 实体所具有的某一特性称为属性
- 码（关键字，Key）
  - 唯一标识实体的（最小的）属性集称为码。例如学号学生实体的码
- 域（Domain）
  - 属性的取值范围称为该属性的域。例如学号的域为8位整数，姓名的域为字符串集合，，性别的域为（男，女）。
- 实体型（Entity Type）
  - 具有相同属性的实体必然具有共同的特征和性质。用实体名及其属性名集合来抽象和刻划同类实体，称为实体型。例如：学生（学号，姓名，性别，出生年月，系，入学时间）
- 实体集（Entity Set）
  - 同型实体的集合称为实体集。例如，全体学生就是一个实体集。
- 联系（Relationship）
  - 在现实世界中，事物内部以及事物之间是有联系的，这些联系在信息世界中反映为实体（型）内部的联系和实体（型）之间的联系。

联系种类

- 1对1联系
  - 定义：若对于实体集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。
- 1对多联系
  - 定义：若对于实体集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具有一对多联系，记为1:n
- 多对多联系
  - 定义：若对于实体集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与实体集B具有多对多联系，记为m:n

表示：E-R图

- 使用长方形来表示实体型，框内写上实体名
- 椭圆型表示实体的属性，并用无向边把实体和属性连接起来
- 用菱形表示实体间的联系，菱形框内写上联系名，用无向边把菱形分别与有关实体相连接，在无向边旁标上联系的类型，若实体之间联系也具有属性，则把属性和菱形也用无向边连接上

组成要素

- 数据结构
  - 数据结构是所研究的对象类型（Object Type）的集合。这些对象是数据库的组成部分。一般可分为两类：一类是与数据类型、内容、性质有关的对象，如网状模型中的数据项、记录，关系模型中的属性、关系等；一类是与数据之间联系有关的对象，如网状模型中的系型（Set Type）等。
- 数据操作
  - 数据操作是指对数据库中各种对象（型）的实例（值）允许执行的操作的集合。数据库主要有检索和更新（插入、删除、修改）两大类操作。
  - 数据结构是对系统静态特性的描述，数据操作是对系统动态特性的描述。

- 数据的约束条件
  - 数据的约束条件是完整性规则的集合。完整性规则是给定的数据模型中数据及其联系所具有的制约和依存规则，用以限定符合数据模型的数据状态以及状态的变化，以保证数据的正确、有效、相容。

基本数据模型类型

层次模型

- 最早使用的一种模型
- 数据结构是一棵有向树
- 特点
  - （1）有且仅有一个结点无双亲，该结点称为根结点。
  - （2）其他结点有且只有一个双亲

网状模型

- 数据结构是一个有向图
- 特点
  - 有一个以上的结点没有双亲
  - 结点可以有多于一个的双亲
- 能表示实体之间的多种复杂联系

关系模型

用二维表格结构来表示实体及实体之间的联系的模型

- 数据结构是一个“二维表框架”组成的集合
- 关系模型概念简单，清晰，用户易懂易用，有严格的数学基础
- 大多数数据库系统都是关系型的

概念

- 关系：一个关系对应于我们平常讲的一张表
- 元组：表中的一行称为一个元组
- 属性：表中的一列称为属性，每列的名称为属性名
- 主码：表中的某个属性组，它们的值唯一的标识一个元组
- 域：属性的取值范围
- 分量：元组中的一个属性值
- 关系模式：对关系的描述，用关系名（属性名1，属性名2，…，属性名n）来表示

特点

（略）

- 是建立在严格数学概念基础上的
- 概念单一
- 存取路径对用户透明，从而有更高的数据独立性和保密性

关系,关系模式,关系模型区别和联系

- 关系:一个关系对应通常说的一张表
- 关系模式:关系的描述
- 关系模型:关系模型由关系数据结构,关系操作集合,关系完整性约束三部分组成.
- 关系和关系模式的区别
    - 关系模式是型,关系是值,关系模式是对关系的描述
    - 关系是关系模式在某一个时刻的状态或者内容,关系模式是静态的,稳定的,而关系是动态的,随时间不断变化的,因为关系操作在不断地更新着数据库中的数据
    - 类似于面向对象程序设计中”类“与”对象“的区别。”关系“是”关系模式“的一个实例，可以把”关系”理解为一张带数据的表，而“关系模式”是这张数据表的表结构。
- 关系模型和关系的区别
    - 关系模型包含关系,关系是关系模型的数据结构,在关系模型中,现实世界的实体以及实体间的各级联系均用单一的结构类型,即关系来表示

DBS三级模式结构

外模式、模式、内模式

- 外模式
  - 又称为用户模式，是数据库用户和数据库系统的接口，是数据库用户的数据视图，是数据库用户可以看见和使用的局部数据的逻辑结构和特征的描述
  - 一个数据库通常都有多个外模式。一个应用程序只能使用一个外模式，但同一外模式可为多个应用程序所用
- 模式
  - 可细分为概念模式和逻辑模式，是所有数据库用户的公共数据视图，是数据库中全部数据的逻辑结构和特征的描述。
  - 一个数据库只有一个模式。模式不但要描述数据的逻辑结构，还要描述数据之间的联系、数据的完整性、安全性要求。
- 内模式
  - 又称为存储模式，是数据库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。
  - 一个数据库只有一个内模式。内模式并不涉及物理记录，也不涉及硬件设备。

为了能够在系统内部实现这三个抽象层次的联系和转换，数据库管理系统在这三级模式之间提供了两层映像：**外模式/模式**映像和**模式/内模式**映像。两层映像保证了数据库系统中的数据能够具有较高的**逻辑独立性和物理独立性。**

二层映象功能

外模式/模式映象和模式/内模式映象

映象

- 是一种对应规则，说明映象双方如何进行转换。

外模式/模式映象

- 作用：把描述局部逻辑结构的外模式与描述全局逻辑结构的模式联系趣来
- 当模式改变时，只要对外模式/模式映象做相应的改变，使外模式保持不变，则以外模式为依据的应用程序不受影响，从而保证了数据与程序之间的逻辑独立性，也就是数据的逻辑独立性

模式/内模式映象

- 作用：把描述全局逻辑结构的模式与描述物理结构的内模式联系起来
- 当内模式改变时，比如存储设备或存储方式有所改变，只要模式/内模式映象做相应的改变，使模式保持不变，则应用程序就不受影响，从而保证了数据与程序之的物理独立性。

DBS组成

- 硬件平台 + 数据库
- 软件
- 人员

2关系DB

关系模型

关系数据结构

- 关系(值)
  - 域
    - 一组具有相同数据类型的值的集合
  - 笛卡尔积
    - 一个域中的元素与另外的域中的元素分别结合构成一个元组
  - 关系
    - 候选码:某一属性组的值能唯一地标识一个元组，而其子集不能，则称该属性组为候选码（candidate key）
    - 主码:若一个关系有多个候选码，则选定其中一个为主码（primary key）

    - 主属性和非主属性：候选码的诸属性称为主属性（prime attribute）
      不包含在任何候选码中的属性称为非主属性（non-prime attribute）或非码属性（non-key attribute）
    - 全码：关系模式的所有属性是这个关系模式的候选码，称为全码（all-key）
- 关系模式(型)
  - 是对关系的描述
- 关系数据库

关系类型

- 基本关系(基本表)
  - 实际存在的表，是实际存储数据的表示，不可再分，必须规范化(也就是达到第一范式1NF)
- 查询表
  - 查询结果对应的表
- 视图表
  - 由基本表或其他视图表导出的表，是虚表，不对应实际存储数据

基本关系性质

① 列是同质的（Homogeneous）

- 每一列中的分量是同一类型的数据，来自同一个域

② 不同的列可出自同一个域

- 其中的每一列称为一个属性
- 不同的属性要给予不同的属性名

③ 列的顺序无所谓

- 列的次序可以任意交换
- 遵循这一性质的数据库产品(如ORACLE)，增加新属性时，永远是插至最后一列。但也有许多关系数据库产品没有遵循这一性质，例如FoxPro仍然区分了属性顺序

④ 任意两个元组的候选码不能完全相同

- 候选码是可以惟一标识一个元组的属性或属性组。若一个关系中的候选码有多个，则选择一个作为主码

⑤ 行的顺序无所谓

- 行的次序可以任意交换
- 遵循这一性质的数据库产品(如ORACLE)，插入一个元组时永远插至最后一行。但也有许多关系数据库产品没有遵循这一性质，例如FoxPro仍然区分了元组的顺序

⑥ 分量必须取原子值

- 每一个分量都必须是不可分的数据项。

关系操作

基本操作

- 选择
- 投影
- 并
- 差
- 笛卡尔积

- 选择σ（Select)(横挑)　　从一个表中把满足条件的元组选出来
  选择是一种单目运算，即对一个关系施加的运算，按给定条件从关系中挑选满足条件的元组组成的集合
  语法格式：σ‹选择条件›（‹关系名›）
- 投影π（Project)(竖挑)　　将需要的属性列出来
  投影操作是单目运算，从关系中挑选指定的属性组成的新关系
  语法格式：π‹属性表›（‹关系名›）
- 并 ∪　　把两个模式相同的元组并起来

- 集合差- 把属于关系A不属于关系B的元组找出来
- 笛卡尔乘积× 两个关系的拼接

关系数据语言

- 关系代数语言
- 关系演算语言
- 双重特性语言(SQL)/结构化查询语言

关系完整性

关系模型中的三类完整性约束 + 规则

- 实体完整性(主键)
- 参照完整性(外键)
- 用户定义的完整性

- 关系模型的完整性规则是对关系的某种约束条件
- 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。

关系代数

重点 - 计算题

- 关系代数是一种抽象的查询语言，用对关系的运算来表达查询
- 关系代数的运算对象是关系，运算结果亦为关系

关系代数运算中的基本运算包括**并**（∪）、**差**（-）、**广义笛卡尔积**（×）、**投影**（π）**和选择**（σ）

传统的集合运算

普通的集合运算

- 并、交、差 + 笛卡尔积

并、交、差

参与集合的两个关系要满足两个条件:

1. 属性个数相同
2. 属性类型要一样

参与并、差操作的两个关系的元组必须限制为同类型的，即具有相同的目，且对应的属性的域相同—并兼容（union compatibility）;

1. **关系操作优先级高于集合操作**;
2. **一元操作（单目）优先级高于二元操作。**

笛卡尔积

1. 结果模式包括进行操作的两个表的所有属性，两张表的**每条元组**之间**两两**拼接
2. 若R和S中有相同的属性名，在这些属性名前加上关系名作为限定词(如：R.B,S.B)，进行区别。

专门的关系运算

选择

选择操作σ（Select)(横挑）从一个表中把满足条件的元组选出来

- 选择是一种单目运算，即对一个关系施加的运算，按给定条件从关系中挑选满足条件的元组组成的集合。
- 语法格式：σ‹**选择条件**›（‹**关系名**›）

投影

投影操作π（Project)(竖挑）将需要的属性列出来

- 投影操作是单目运算，从关系中挑选指定的属性组成的新关系。

- 语法格式：**π‹属性表› （‹关系名›）**

连接

R∞‹连接条件›S = σ‹连接条件› （R * S）

- 解释：关系R和关系S在该条件的连接操作等于关系R和关系S先做笛卡尔乘积，再按照该条件做选择操作

等值连接：

- 一种特殊的条件连接，连接条件只有等值的条件
- 结果模式和笛卡尔乘积的模式类似，把等值的属性去掉一列

自然连接：两张表在所有的公共属性上做等值连接

- 步骤：
  1. 作R x S（笛卡儿积）
  2. 在R x S上选择同名的属性组
  3. 去掉重复属性
- 一般的连接操作是从**行**的角度进行运算，但自然连接还需要**取消重复列**，所以是同时**从行和列**的角度进行运算

除

A/B = {‹x› | ∃‹x,y› ∈A, ∀‹y› ∈B}

计算思路：比如对于A/B来说，我们要找在关系A中跟关系B中所有y值都有联系的x值。【也就是找出在关系B中没有一个y值与A的是没有联系的。否定的否定，就是找出B中的列的属性在A中都有关联的属性，B中列的属性不写.】

- 先在A中找不满足除法条件的x
  - 先把A做一个投影，投影到x属性
  - 将投影结果和关系B做一个笛卡尔乘积
  - 用笛卡尔乘积结果-A
  - 对上一步结果做一个投影，投影到x
- 把关系A所有的x值减去所有不满足条件的x值

- 把关系A投影到x
- 投影结果减去上一操作找到的所有不满足条件的x值

关系代数运算小册

- 并Union（∪）
  - R和S的并，R∪S，是在R或S或两者中的元素的集合
  - 一个元素在并集中只出现一次
  - R和S必须同类型（属性集相同、次序相同，但属性名可以不同）
- 交Intersect（∩）
  - R和S的交，R∩S，是在R和S中都存在的元素的集合
  - 一个元素在交集中只出现一次
  - R和S必须同类型（属性集相同、次序相同，但属性名可以不同）
- 差Minus（-）
  - R和S的差，R-S，是在R中而不在S中的元素的集合
  - R和S必须同类型（属性集相同、次序相同，但属性名可以不同）
- 投影Projection(π)
  - 从关系R中选择若干属性组成新的关系
  - πA1,A2,…,An(R),表示从R中选择属性集A1,A2,…,An组成新的关系
  - 列的运算
  - 投影运算的结果中,也要去除可能的重复元组
- 广义笛卡儿积(×)
  - 关系R、S的广义笛卡儿积是两个关系的元组对的集合所组成的新关系
  - R×S：
    - 属性是R和S的组合（有重复）
    - 元组是R和S所有元组的可能组合
    - 是R、S的无条件连接，使任意两个关系的信息能组合在一起
- 选择Selection(σ)
  - 从关系R中选择符合条件的元组构成新的关系
  - σC(R),表示从R中选择满足条件(使逻辑表达式C为真)的元组

    - 行的运算
- 条件连接(θ)
    - 从R×S的结果集中，选取在指定的属性集上满足AθB条件的元组，组成新的关系
    - θ是一个关于属性集的比较运算符
    - θ为“=”的连接运算称为等值连接。
- 自然连接
    - 从R×S的结果集中，选取在某些公共属性上具有相同值的元组，组成新的关系
    - R、S的公共属性
        - 属性集的交集（名称及类型相同）
    - 公共属性在结果中只出现一次
    - 等值连接
- 关系代数–除(÷)
    - 给定关系R(X,Y)和S(Y,Z)，其中X，Y，Z为属性组。R中的Y与S中的Y可以有不同的属性名，但必须出自相同的域集。R与S的除运算得到一个新的关系P(X)，P是R中满足下列条件的元组在X属性列上的投影：元组在X上分量值x的象集Yx包含S在Y上投影的集合。
    - R÷S = {tr[X]| tr∈R∧πy (S)Yx}
    - 其中Yx为x在R中的象集，x=tr[X]。
    - 例子

3SQL

SQL:结构化查询语言（Structured Query Language）

见个人MYSQL笔记本：Mysql相关

定义

（略）

非过程化语言

- SQL语言进行数据库操作时，只需要提出“做什么”，不需要指明“怎么做”。“怎么做”是由DBMS来完成

SQL的形式

- 交互式SQL
  - 一般DBMS都提供联机交互工具
  - 用户可直接键入SQL命令对数据库进行操作
  - 由DBMS来进行解释
- 嵌入式SQL
  - 能将SQL语句嵌入到高级语言（宿主语言）
  - 使应用程序充分利用SQL访问数据库的能力、宿主语言的过程处理能力
  - 一般需要预编译，将嵌入的SQL语句转化为宿主语言编译器能处理的语句

三级模式结构对应的关系

- 外模式->视图
  - 视图是从一个或几个基本表导出的表。它本身不独立存储在数据库中，即数据库中**只存放视图的定义而不存放视图对应的数据。**这些数据仍存放在导出视图的基本表中，因此视图是一个**虚表。**视图在概念上与基本表等同，用户可以在视图上再定义视图。
- 模式->基本表
  - 基本表是本身**独立存在**的表，在关系数据库管理系统中**一个关系就对应一个基本表。**一个或多个基本表对应一个存储文件，一个表可以带若干索引，索引也存放在存储文件中。
- 内模式->存储文件
  - 存储文件的逻辑结构组成了关系数据库的内模式，存储文件的物理结构对最终用户是隐蔽的。

组成

- 数据定义语言（DDL，Data Definition Language）

  数据定义语言是指用来定义和管理数据库以及数据库中的各种对象的语句，这些语句包括CREATE、ALTER和DROP等语句。在SQL Server中，数据库对象包括表、视图、触发器、存储过程、规则、缺省、用户自定义的数据类型等。这些对象的创建、修改和删除等都可以通过使用CREATE、ALTER、DROP等语句来完成。

  - 常见数据类型

        - 字符型：
            - 定长字符型 char(n) 由于是定长，所以速度快
            - 变长字符型 varchar(n)
        - 数值型：
            - 整型 int(或integer) -231~+231
            - 短整型 smallint -215~+215的
            - 浮点型 real、float、double
            - 数值型 numeric (p [,d])
        - 日期 / 时间型：
            - DateTime
        - 文本和图像型
            - Text：存放大量文本数据。在SQLServer中，Text对象实际为一指针
            - Image：存放图形数据
- 数据操纵语言（DML，Data Manipulation Language）
    - 数据操纵语言是指用来查询、添加、修改和删除数据库中数据的语句，这些语句包括SELECT、INSERT、UPDATE、DELETE等。在默认情况下，只有sysadmin、dbcreator、db_owner或db_datawriter等角色的成员才有权利执行数据操纵语言。
- 数据控制语言（DCL，Data Control Language）
    - 数据控制语言（DCL）是用来设置或者更改数据库用户或角色权限的语句，这些语句包括GRANT、REVOKE 、DENY等语句，在默认状态下，只有sysadmin、dbcreator、db_owner或db_securityadmin等角色的成员才有权利执行数据控制语言。

SQL语句

建立表结构 Create

- 定义基本表的语句格式：
- CREATE TABLE <表名>(<列定义>[{,<列定义>,<表约束>}])
- 表名：
- 列定义：列名、列数据类型、长度、是否允许空值等。
- 定义完整性约束：列约束和表约束
- [CONSTRAINT<约束名>] <约束定义>

删除表结构 Drop

- 用SQL删除关系（表）
  - 将整个关系模式（表结构）彻底删除
  - 表中的数据也将被删除

修改表结构 Alter

- 增加表中的属性
  - 向已经存在的表中添加属性
  - `allow null` （新添加的属性要允许为空）
  - 已有的元组中该属性的值被置为Null
- 修改表中的某属性(某列)
  - 修改属性类型或精度
- 删除表中的某属性(某列)
  - 去除属性及相应的数据

向表中添加数据(Insert)

- 数据添加
  - 用SQL的插入语句，向数据库表中添加数据
  - 按关系模式的属性顺序
  - 按指定的属性顺序，也可以只添加部分属性（非Null属性为必需）

数据删除（Delete）

- 只能对整个元组操作，不能只删除某些属性上的值
- 只能对一个关系起作用，若要从多个关系中删除元组，则必须对每个关系分别执行删除命令
- 删除单个元组
- 删除多个元组
- 删除整个关系中的所有数据

数据更新（Update）

- 改变符合条件的某个（某些）元组的属性值

数据查询（Select）

数据查询是数据库应用的核心功能

Select子句—重复元组

- SQL具有包的特性
- Select 子句的缺省情况是保留重复元组（ ALL ），可用 Distinct 去除重复元组
- 去除重复元组:费时
- 需要临时表的支持

Select子句— *与属性列表

- 星号 * 表示所有属性
  - 星号 * ：按关系模式中属性的顺序排列
  - 显式列出属性名：按用户顺序排列

Select子句—更名

- 为结果集中的某个属性改名
- 使结果集更具可读性

Where 子句

- 查询满足指定条件的元组可以通过Where子句来实现
- 使where子句中的逻辑表达式返回True值的元组，是符合要求的元组，将被选择出来
- Where 子句—运算符
  - 比较：<、<=、>、>=、=、<> 等
  - 确定范围：
    - Between A and B、Not Between A and B
  - 确定集合：IN、NOT IN
  - 字符匹配：LIKE, NOT LIKE

    - 空值：IS NULL、IS NOT NULL
    - 多重条件：AND、OR、NOT
- Where 子句—Like
    - 字符匹配：Like、Not Like
    - 通配符
    - % — 匹配任意字符串
    - _ — 匹配任意一个字符
    - 大小写敏感
- Where 子句—转义符 escape

From 子句

- 列出将被查询的关系（表）
- From 子句—元组变量
    - 为 From 子句中的关系定义元组变量
    - 方便关系名的引用
- 连接子句
    - 内连接
        - 内连接是指包括符合条件的每个表的记录，也称之为全记录操作。
    - 外连接
        - 外连接是指把两个表分为左右两个表。右外连接是指连接满足条件右侧表的全部记录。左外连接是指连接满足条件左侧表的全部记录。全外连接是指连接满足条件表的全部记录。
        - 左外连接
        - 右外连接
        - 全外连接

Order By子句

- 指定结果集中元组的排列次序
- 耗时
- ASC升序（缺省）、DESC降序

子查询（Subquery）

- 子查询是嵌套在另一查询中的 Select-From-Where 表达式（Where/Having）
- SQL允许多层嵌套，由内而外地进行分析，子查询的结果作为父查询的查找条件
- 可以用多个简单查询来构成复杂查询，以增强SQL的查询能力
- 子查询中不使用 Order By 子句，Order By子句只能对最终查询结果进行排序
- 子查询—单值比较
  - 返回单值的子查询，只返回一行一列
  - 父查询与单值子查询之间用比较运算符进行连接
    - 运算符：>、>=、=、<=、<、 <>
- 子查询—多值
  - 子查询返回多行一列
  - 运算符：In、All、Some(或Any)、Exists
    - 子查询—多值成员In
      - 若值与子查询返回集中的某一个相等，则返回true
        - IN 被用来测试多值中的成员
    - 子查询—多值比较 ALL
      - 父查询与多值子查询之间的比较用All来连接
      - 值s比子查询返回集R中的每个都大时，s>All R 为True
      - All表示所有
      - ```
        all、< all、<=all、>=all、<> all
        ```
      - <> all 等价于 not in
    - 子查询—多值比较Some/Any
      - 父查询与多值子查询之间的比较需用Some/Any来连接
      - 值s比子查询返回集R中的某一个都大时返回 Ture
        - s > Some R为True 或
        - s > Any R为True
      - Some(早期用Any)表示某一个（任意一个）
      - ```
        some、< some、<=some、>=some、<> some
        ```
      - = some 等价于 in、<> some 不等价于 not in
- 子查询—存在判断Exists
  - Exists + 子查询用来判断该子查询是否返回元组
  - 当子查询的结果集非空时，Exists为True

- 当子查询的结果集为空时，Exists为False
- 不关心子查询的具体内容，因此用 Select *
- 具有外部引用的子查询，称为相关子查询（Correlated Queries）
- 外层元组的属性作为内层子查询的条件

聚合函数

- 把一列中的值进行聚合运算，返回单值的函数
- 五个预定义的聚合函数
  - 平均值：Avg( )
  - 总和： Sum( )
  - 最小值：Min( )
  - 最大值：Max( )
  - 计数： Count( ) 返回所选列中不为NULL的数
- Group By
  - 将查询结果集按某一列或多列的值分组，值相等的为一组，一个分组以一个元组的形式出现
  - 只有出现在Group By子句中的属性，才可出现在Select子句中
- Having
  - 针对聚合函数的结果值进行筛选（选择），它作用于分组计算结果集
  - 跟在Group By子句的后面。
- Having 与 Where的区别
  - Where 决定哪些元组被选择参加运算，作用于关系中的元组
  - Having 决定哪些分组符合要求，作用于分组
  - 聚合函数的条件关系必须用Having，Where中不应出现聚合函数
- 聚合函数对Null的处理
  - Count：不计
  - Sum：不将其计入
  - Avg：具有 Null 的元组不参与
  - Max / Min：不参与

视图

视图是从一个或者多个表或视图中导出的表，其结构和数据是建立在对表的查询基础上的。和真实的表一样，视图也包括几个被定义的数据列和多个数据行，但从本质上讲，这些数据列和数据行来源于其所引用的表。因此，视图不是真实存在的基础表而是一个虚拟表，视图所对应的数据并不实际地以视图结构存储在数据库中，而是存储在视图所引用的表中。

- 视图定义只在数据字典中存定义，不存数据并且不会执行基表的查询

CRUD

定义视图

CREATE VIEW <Name>;

视图建立在基本表上

```
CREATE VIEW IS_Student
AS
SELECT Sno，Sname，Sage
FROM Student
WHERE Sdept=IS;
```

视图建立在已经定义好的视图上

FROM 视图即可

删除视图

DROP VIEW<视图名>[CASCADE];

删除视图的时候,如果该视图上还导出了其他的视图,那么不能删除,需要在后面加CASCADE,即级联操作,进而把该视图和它导出的所有视图删除

查询视图

视图消解:(p124)从数据字典中取出视图的定义，把定义中的子查询和户的查询结合起来，转换成等价的对基本表的查询，然后再执行修正了的查询。这一转发过程称为视图消解（view resolution）

实际上就是把视图的中定义的子查询查询出来的结果,再在查询视图的语句上根据上面的结果写出视图消解的后的转换语句

更新视图

自动转换为对基本表的更新

作用

- 简化用户操作
- 多角度看待同一数据
- 对重构数据库提供逻辑独立性
- 保护机密数据
- 优化查询

索引

某个表中一列或者若干列值的集合和相应的指向表中物理标识这些值的数据页的逻辑指针List

CRUD

Link

创建索引

```
-- 创建普通索引
CREATE INDEX index_name ON table_name(col_name);

-- 创建唯一索引
CREATE UNIQUE INDEX index_name ON table_name(col_name);

-- 创建普通组合索引
CREATE INDEX index_name ON table_name(col_name_1,col_name_2);

-- 创建唯一组合索引
CREATE UNIQUE INDEX index_name ON table_name(col_name_1,col_name_2);
```

修改表结构创建索引

```sql
ALTER TABLE table_name ADD INDEX index_name(col_name);
```

创建表时直接指定索引

```sql
CREATE TABLE table_name (
    ID INT NOT NULL,
    col_name VARCHAR (16) NOT NULL,
    INDEX index_name (col_name)
);
```

删除索引

```sql
-- 直接删除索引
DROP INDEX index_name ON table_name;

-- 修改表结构删除索引
ALTER TABLE table_name DROP INDEX index_name;
```

其他

```sql
-- 查看表结构
desc table_name;

-- 查看生成表的SQL
show create table table_name;

-- 查看索引信息（包括索引结构等）
show index from  table_name;

-- 查看SQL执行时间（精确到小数点后8位）
set profiling = 1;
SQL...
show profiles;
```

作用

略

- 通过创建唯一索引，可以保证数据记录的唯一性。
- 可以大大加快数据检索速度。
- 可以加速表与表之间的连接，这一点在实现数据的参照完整性方面有特别的意义。
- 在使用ORDER BY和GROUP BY子句中进行检索数据时，可以显著减少查询中分组和排序的时间。
- 使用索引可以在检索数据的过程中使用优化隐藏器，提高系统性能

约束

略

- 主键约束 (primary key constraint)
- 唯一性约束 (unique constraint)
- 检查约束 (check constraint)
- 缺省约束 (default constraint)
- 外键约束 (foreign key constraint)

SQL SERVER权限管理

（自己补充内容，无关）

SQL Server权限管理策略

- 安全帐户认证
    - 安全帐户认证是用来确认登录SQL Server的用户的登录帐号和密码的正确性，由此来验证其是否具有连接SQL Server的权限。 SQL Server 2000提供了两种确认用户的认证模式：
        - (一) Windows NT认证模式。
            - SQL Server数据库系统通常运行在Windows NT服务器平台上，而NT作为网络操作系统，本身就具备管理登录、验证用户合法性的能力，因此Windows NT认证模式正是利用了这一用户安全性和帐号管理的机制，允许SQL Server也可以使用NT的用户名和口令。在这种模式下，用户只需要通过Windows NT的认证，就可以连接到SQL Server，而SQL Server本身也就不需要管理一套登录数据。

- （二）混合认证模式。
  - 混合认证模式允许用户使用Windows NT安全性或SQL Server安全性连接到SQL Server，这就意味着用户可以使用他的帐号登录到Windows NT，或者使用他的登录名登录到SQL Server系统。NT的用户既可以使用NT认证，也可以使用SQL Server认证

- 访问许可确认
  - 但是通过认证阶段并不代表用户能够访问SQL Server中的数据，同时他还必须通过许可确认。用户只有在具有访问数据库的权限之后，才能够对服务器上的数据库进行权限许可下的各种操作，这种用户访问数据库权限的设置是通过用户帐号来实现的。

用户权限管理

- 服务器登录帐号和用户帐号管理
  - 1.利用企业管理器创建、管理SQL Server登录帐号
    - （1）打开企业管理器，单击需要登录的服务器左边的“+”号，然后展开安全性文件夹。
    - （2）用右键单击登录（login）图标，从快捷菜单中选择新建登录（new login）选项，则出现SQL Server登录属性–新建登录对话框，如图6-2所示。
    - （3）在名称编辑框中输入登录名，在身份验证选项栏中选择新建的用户帐号是Windows NT认证模式，还是SQL Server认证模式。
    - （4）选择服务器角色页框。在服务器角色列表框中，列出了系统的固定服务器角色。
    - （5）选择用户映射页框。上面的列表框列出了该帐号可以访问的数据库，单击数据库左边的复选框，表示该用户可以访问相应的数据库以及该帐号在数据库中的用户名。
    - （6）设置完成后，单击“确定”按钮即可完成登录帐号的创建。
  - 使用SQL 语句创建登录帐号
  - 2.用户帐号管理
    - 在数据库中，一个用户或工作组取得合法的登录帐号，只表明该帐号通过了Windows NT认证或者SQL Server认证，但不能表明其可以对数据库数据和数据库对象进行某种或者某些操作，只有当他同时拥有了用户权限后，才能够访问数据库。
    - 利用企业管理器可以授予SQL Server登录访问数据库的许可权限。使用它可创建一个新数据库用户帐号
- 许可（权限）管理
  - 许可用来指定授权用户可以使用的数据库对象和这些授权用户可以对这些数据库对象执行的操作。用户在登录到SQL Server之后，其用户帐号所归属的NT组或角色所被赋予的许可（权限）决定了该用户能够对哪些数据库对象执行哪种操作以及能够访问、修改哪些数据。在每个数据库中用户的许可独立于用户帐号和用户在数据库中的角色，每个数据库都有自己独立的许可系统，在SQL Server中包括三种类型的许可：即对象许可、语句许可和预定义许可。
    - 三种许可类型
      - 1、对象许可

        - 表示对特定的数据库对象，即表、视图、字段和存储过程的操作许可，它决定了能对表、视图等数据库对象执行哪些操作。
    - 2、语句许可
        - 表示对数据库的操作许可，也就是说，创建数据库或者创建数据库中的其它内容所需要的许可类型称为语句许可。
    - 3、预定义许可
        - 是指系统安装以后有些用户和角色不必授权就有的许可。
- 角色管理
    - 角色是SQL Server 7.0版本引进的新概念，它代替了以前版本中组的概念。利用角色，SQL Server管理者可以将某些用户设置为某一角色，这样只对角色进行权限设置便可以实现对所有用户权限的设置，大大减少了管理员的工作量。SQL Server提供了用户通常管理工作的预定义服务器角色和数据库角色。
        - 1、服务器角色
            - 服务器角色是指根据SQL Server的管理任务，以及这些任务相对的重要性等级来把具有SQL Server管理职能的用户划分为不同的用户组，每一组所具有的管理SQL Server的权限都是SQL Server内置的，即不能对其进行添加、修改和删除，只能向其中加入用户或者其他角色。
            - 几种常用的固定服务器角色
                - 系统管理员：拥有SQL Server所有的权限许可。
                - 服务器管理员：管理SQL Server服务器端的设置。
                - 磁盘管理员：管理磁盘文件。
                - 进程管理员：管理SQL Server系统进程。
                - 安全管理员：管理和审核SQL Server系统登录。
                - 安装管理员：增加、删除连接服务器，建立数据库复制以及管理扩展存储过程。
                - 数据库创建者：创建数据库，并对数据库进行修改。
        - 2、数据库角色
            - 数据库角色是为某一用户或某一组用户授予不同级别的管理或访问数据库以及数据库对象的权限，这些权限是数据库专有的，并且还可以使一个用户具有属于同一数据库的多个角色。SQL Server提供了两种类型的数据库角色：即固定的数据库角色和用户自定义的数据库角色。
            - （1）固定的数据库角色
                - public：维护全部默认许可。
                - db_owner：数据库的所有者，可以对所拥有的数据库执行任何操作。
                - db_accessadmin：可以增加或者删除数据库用户、工作组和角色。
                - db_addladmin：可以增加、删除和修改数据库中的任何对象。
                - db_securityadmin：执行语句许可和对象许可。

- db_backupoperator：可以备份和恢复数据库。
- db_datareader：能且仅能对数据库中的任何表执行select操作，从而读取所有表的信息。
- db_datawriter：能够增加、修改和删除表中的数据，但不能进行select操作。
- db_denydatareader：不能读取数据库中任何表中的数据。
- db_denydatawriter：不能对数据库中的任何表执行增加、修改和删除数据操作。

- （2）用户自定义角色
  - 创建用户定义的数据库角色就是创建一组用户，这些用户具有相同的一组许可。如果一组用户需要执行在SQL Server中指定的一组操作并且不存在对应的Windows NT组，或者没有管理Windows NT用户帐号的许可，就可以在数据库中建立一个用户自定义的数据库角色。用户自定义的数据库角色有两种类型：即标准角色和应用程序角色。

Transaction_SQL 语句

- 赋权语句—Grant
- 收回权限—Revoke
- 收回权限—Deny

4安全性

方法

1. 用户标示鉴定
2. 存取控制
3. 审计
4. 数据加密
5. 视图的保护

存储控制

自主存储控制

通过SQL的GRANT语句和REVOKE语句来实现

- 用户对不同的数据对象有不同的存取权限
- 不同的用户对同一对象也有不同的权限
- 用户还可以将其拥有的存取权限转授给其他用户

Grant授予

把对表SC的INSERT权限授予U5用户，并允许将此权限再授予其他用户。

```
GRANT INSERT
ON TABLE SC
TO U5
WITH GRANT OPTION;
-- 执行此SQL语句后，U5不仅拥有了对表SC的INSERT权限，还可以传播此权限，即由U5用户发上述GRANT命令给其他用户。例如U5可以将此权限授予U6（例4.6）。
```

Revoke收回

把用户U5对SC表的INSERT权限收回

```
REVOKE INSERT
ON TABLE SC
FROM U5 CASCADE;
-- 将用户U5的INSERT权限收回的同时，级联（CASCADE）收回了U6和U7的INSERT权限，否则系统将拒绝执行该命令。
```

强制存储控制

- 在强制存取控制中，数据库管理系统所管理的全部实体被分为主体和客体两大类。

1. 主体是系统中的活动实体，既包括数据库管理系统所管理的实际用户，也包括代表用户的各进程。
2. 客体是系统中的被动实体，是受主体操纵的，包括文件、基本表、索引、视图等

- 强制存取控制是**对数据本身进行密级标记**，无论数据如何复制，标记与数据是一个不可分的整体，只有符合密级标记要求的用户才可以操纵数据，从而提供了更高级别的安全性。

- 实现MAC时首先要实现DAC
- 规则
  - 仅当主体的许可证级别**大于或等于**客体的密级时，该主体才能**读**取相应的客体。
  - 仅当主题的许可证级别**等于**客体的密级时，该主体才能**写**相应的客体。
- 规则的共同点
  - **禁止**了拥有**高**许可证级别的主体更新**低密级**的数据对象

视图机制

通过视图机制把要保密的数据对无权存取的用户隐藏起来，从而自动对数据提供一定程度的安全保护。

就是通过视图对不同的用户授予不同的权限

审计

记录操作到日志

数据加密

5完整性

实体完整性

主属性

关系模型的实体完整性在`CREATE TABLE`中用**`PRIMARY KEY`**定义

1. 列级约束条件：定义在属性的后面
    - ```
      Sno CHAR(9) PRIMARY KEY
      ```
2. 表级约束条件：单独成一行，使用语句单独定义
    - ```
      PRIMARY KEY(Sno)
      ```

- 实体完整性
    - ```
      create table student
      (Sno char(9) primary key,
       ...);

       //表级
       create table student
       (  Sno char(9),
          ....,
           primary key(Sno)
       )
      ```
- 参照完整性
    - ```
      create table sc
      (   sno char(9) not null,
          foreign key(sno) references student(Sno)
          on delete cascade
          on update cascade
      )
      ```
- 用户定义完整性* 属性上 not null, unique, check(...)
    - 元组：check(...)

参照完整性

外码

关系模型的参照完整性在CREATE TABLE中用FOREIGN KEY短语定义哪些列为**外码**，用REFERENCES短语指明这些外码**参照哪些表的主码**

```
CREATE TABLESC
(Sno CHAR（9）NOT NULL,
  Cno CHAR（4）NOT NULL,
  Grade SMALLINT,
  PRIMARY KEY（Sno，Cno）,-- 在表级定义实体完整性
  FOREIGN KEY（Sno）REFERENCES Student（Sno）,-- 在表级定义参照完整性
  FOREIGN   KEY（Cno）REFERENCES Course（Cno）-- 在表级定义参照完整性
  )
```

用户定义完整性

1. 属性上的约束条件
    - 列值非空（NOTNULL）
    - 列值唯一（UNIQUE）
    - 检查列值是否满足一个条件表达式（CHECK短语）
2. 元组上的约束条件

当学生的性别是男时，其名字不能以Ms.打头

```
CREATE TABLE Student
(Sno CHAR（9）,
  Sname CHAR（8）NOT NULL,
  Ssex CHAR（2）,
  Sage SMALLINT,
  Sdept CHAR（20）,
  PRIMARY KEY（Sno）,
  CHECK（Ssex='女'OR Sname NOT LIKE'Ms.%'）
);
-- 定义了元组中Sname和Ssex两个属性值之间的约束条件
```

完整性约束命名子句

略

Constraint 条件名 约束条件

断言

略

触发器

trigger

CRUD

Link

分类

DDL

> 主要包括 create alter drop

DML

> 主要包括 insert update delete

示例

insert 触发器的创建

在 STUMS 数据库的教师表上创建一个名为 js_insert_trigger 的触发器，当执行 INSERT 操作时，该触发器被触发，提示 “ 禁止插入记录！” 。

```
CREATE TRIGGER js_insert_trigger ON 教师
FOR INSERT
AS
BEGIN
PRINT('禁止插入记录！')
ROLLBACK TRANSACTION
END
GO
```

delete 触发器的创建

在 STUMS 数据库的教师表上创建一个名为 js_delete_trigger 的触发器，当执行 DELETE 操作时，该触发器被触发，提示“ 禁止删除记录！”。
代码如下

```
USE STUMS
GO
CREATE TRIGGER js_delete_trigger ON 教师
FOR DELETE
AS
BEGIN
PRINT('禁止删除记录！')
ROLLBACK TRANSACTION
END
GO
```

多表级联插入触发器

在 STUMS 数据库的学生基本信息表上创建一个名为 xs_insert_trigger 的触发器，当在学生基本信息表中插入记录时，将该记录中的学号自动插入 Student 表。

```
USE STUMS
GO
CREATE TRIGGER xs_insert_trigger ON 学生基本信息
FOR INSERT
AS
DECLARE @XH CHAR(9)  /*定义局部变量*/
SELECT @XH = 学号 FROM INSERTED  /*从INSERTED表中取出学号赋给变量@XH */
INSERT Student(学号)
VALUES(@XH)   /*将变量@XH的值插入到选课表*/
GO
```

下略

删除

```
Drop Trigger
```

6关系数据理论

好关系模式

一个好的模式应当不会发生插入异常、删除异常和更新异常，数据冗余应尽可能少

- 插入异常：数据本该插入却插入不了，可能由于实体完整性的约束导致插入失败
- 删除异常：数据本该留下，却在删除其他数据的同时删除了该数据
- 更新异常：更新某个数据导致每个数据都要更新
- 数据冗余：某些数据存在重复

规范化

研究如何把一个不好的关系模式转化为好的关系模式的理论

用几个简单的关系去取代原来结构复杂的关系的过程叫做关系规范化.

- 基本思想：逐步消除数据依赖中不适合的部分
- 模式设计原则：“**一事一地**”
- 规范化的实质：概念的**单一化**
- **关系模式的规范化过程是通过对关系模式的分解来实现的，即把低一级的关系模式分解为若干个高一级的关系模式**

函数依赖

类似于变量之间的单值函数关系

Y=F(X)，其中自变量X的值，决定一个唯一的函数值Y

在一个关系模式里的属性，由于它在不同元组里属性值可能不同，由此可以把关系中的属性看作变量

一个属性与另一个属性在取值上可能存在制约关系

函数依赖就是属性间的逻辑依赖关系

定义1 设R(U)是一个关系模式,U是R的属性集合,X和Y是U的子集.对于R(U)的任何一个可能的关系r,如果r中不存在两个元组,它们在X上的属性值相同,而在Y上的属性值不同,则称X函数决定Y,或Y函数依赖于X,记作:X Y.

X通常称为“决定因素”

几点说明

1. 函数依赖是语义范畴的概念.它反映了一种语义完整性约束,只能根据语义来确定一个函数依赖.
2. 函数依赖是指关系R模式的所有关系元组均应满足的约束条件,而不是关系模式中的某个或某些元组满足的约束条件
3. 函数依赖与属性间的联系类型有关
   - (1)若属性X和Y之间有“一对一”的联系,
   - (2)若属性X和Y之间有“多对一”的联系,
   - (3)若属性X和Y之间有“多对多”的联系,
4. 如果X Y,并且Y不是X的子集,则称X Y是非平凡的函数依赖;如果Y是X的子集,则称X Y是平凡的函数依赖;

完全函数依赖

- 在一张表中，若 X → Y，且对于 X 的任何一个真子集（假如属性组 X 包含超过一个属性的话），X ’ → Y 不成立，那么我们称 Y 对于 X 完全函数依赖,用F表示

部分函数依赖

- Y 函数依赖于 X，但同时 Y 并不完全函数依赖于 X，那么我们就称 Y 部分函数依赖于 X,用P表示

传递函数依赖

- Z 函数依赖于 Y，且 Y 函数依赖于 X ,Y 不包含于 X，且 X 不函数依赖于 Y(如果这样,就会变成直接传递依赖),就称 Z 传递函数依赖于 X,用T表示

平凡函数依赖

- 当Y包含于X时，称X→Y为平凡的函数依赖

非平凡函数依赖

- 当Y不包含于X时，称X→Y为非平凡的函数依赖

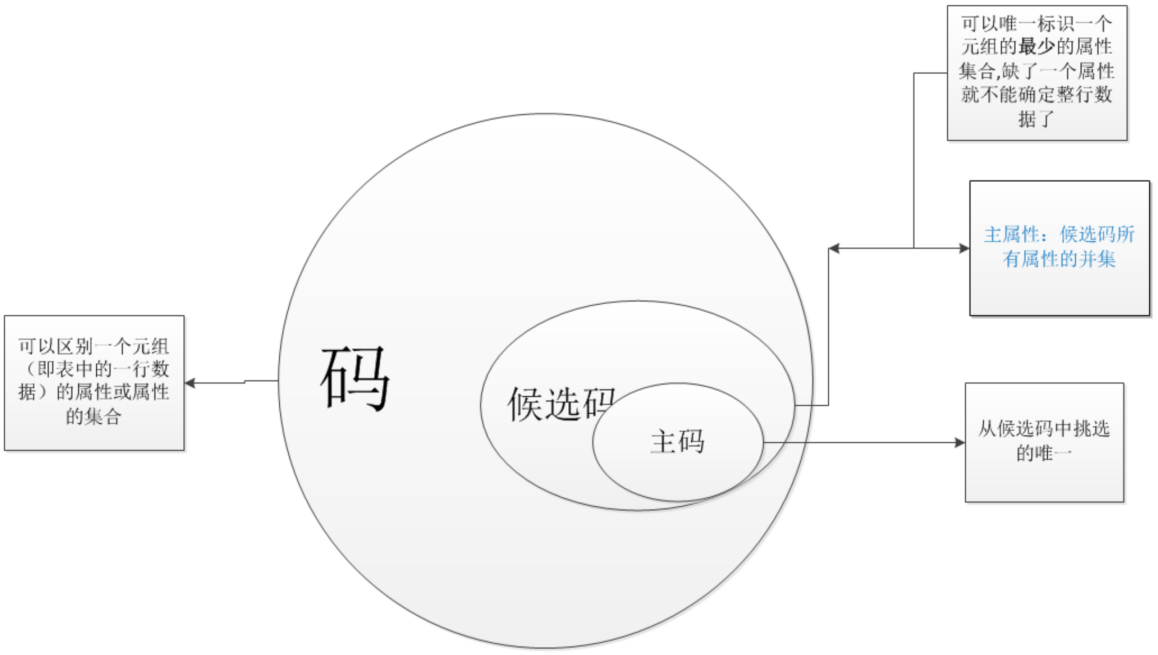
```
当X不包含Y时候，X -> Y是 非平凡
```

码

候选码的两个性质

1. 标识的唯一性：对于R(U)中的每一元组,K的值确定后,该元组就相应确定了.
2. 无冗余性：K是属性组的情况下,K的任何一部分都不能唯一标识该元组(定义中的完全函数依赖的意义)

1. **候选码**:某一个属性组的值能够**唯一的标识**一个元组,而其**子集不能**,则称该属性组为候选码
2. **主码**:若一个关系中有多个候选码,则**选定其中的一个**为主码
3. **主属性**:候选码**属性组中的各个属性**称为主属性
4. **非主属性**:**不包含在候选码中的属性**称为非主属性
5. **全码**:关系模式的**所有属性**是这个关系的候选码,则称该属性组为全码
6. **码**：将主码和候选码简称为码
7. **决定因素**：箭头左边的属性．例如：A→B，由A能够推出B，A就是决定因素



候选码：关系中的一个属性或者属性组，能够唯一标识一个元组，且它的真子集不能唯一标识元组。

主码：从所有候选码中选择一个，作为主码。例如：学生关系（学号，身份证号，姓名，院系，专业，性别，生日），有两个候选码：【学号】和【身份证号】，我们可以选择学号为主码，也可以选择身份证号为主码（当然，一般还是选择学号为主码）。

属性：上例中：学号、身份证号、姓名。。。都是学生的属性。

主属性：候选码的中的一个属性。上例中的学号、身份证号都可以称为主属性。选课（学号，课程号），此关系的候选码只有一个，为：【学号、课程号】，故主属性有：学号、课程号。

解候选码

1. **求候选码**：

   考题，求出下列F的候选码

   R<U,F>,U=(A,B,C,D,E,G),F={AB-->C,CD-->E,E-->A.A-->G},求候选码。

   AB->C
   CD->E
   E->A
   A->G

   step1:只出现在左边的一定是候选码
   step2:只出现在右边的一定不是候选码
   step3:左右都出现的不一定
   step4：左右都不出现的一定是候选码

   一定是候选码的 BD
   一定不是候选码的 G

   可能：ACE

   求BD的闭包，表示为(BD)+ = BD 不是全体，那么把可能的添加进去

   闭包：BD的闭包 是指由BD能推出来的所有属性

   (ABD)+ = ABCDEG
   (BDC)+ = ABCDEG
   (BDE)+ = ABCDEG

   最终：候选码为ABD,BDC,BDE 有三个候选码

   CSDN @心意310

2. **依次对照各种范式的条件定位到属于哪种范式**.
3. **按要求将范式分解到更高级别的范式**.
   - 如果是存在部分函数依赖，就将引起部分函数依赖的那部分拿出单独建一个表.
   - 如果是存在传递函数依赖，就将引起传递函数依赖的那部分拿出单独建一个表.

范式

规范化理论是围绕着范式建立的.

满足不同程度要求的约束集则称为不同的范式

如果一个关系满足某个指定的约束集,则称它属于某个特定的范式.

较高层次的范式比较低层次的范式具有“更合乎要求的性质”

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**

如果一个关系满足某个范式要求,则它也会满足较其级别低的所有范式的要求

- 第一范式：所有的属性不可再分
- 第二范式：在第一范式的基础上，不存在非主属性对码的部分函数依赖
- 第三范式：在第二范式的基础上，不存在非主属性对码的传递函数依赖
  - 因为二，三范式都是针对非主属性的，所以全码可以直接定位到第三范式
- BCNF范式：在第三范式的基础上，不存在主属性对码的部分和传递函数的依赖
  - 判断第三范式是否为BCNF范式：如果每一个决定因素都是包含码，那么就是BCNF

1NF

在关系模式R中的每一个具体关系r中,如果每个属性值都是不可再分的最小数据单位,则称R是第一范式的关系,记作$R \in 1NF$

2NF

若关系模式$R \in 1NF$,且每个非主属性都完全依赖于R的任意候选码,则关系模式R属于第二范式,记作$R \in 2NF$

3NF

若关系模式$R \in 2NF$,且每个非主属性都不传递依赖于R的任意候选码,则$R \in 3NF$

- 从2NF关系中,消除非主属性对码的传递依赖函数而获得3NF关系
- $R \in 3NF$,则每个非主属性既不部分依赖,也不传递依赖于R的任何候选码.

BCNF

若R∈1NF，且R中每个决定因素都是候选码，则R ∈BCNF .

- 满足BCNF的关系将消除任何属性对候选码的部分依赖与传递依赖

  应用BCNF定义时，可直接判断1NF是否属于BCNF

范式判断

1. 找出数据表中所有的**候选码。**
2. 根据第一步所得到的码，找出所有的**主属性。**
3. 数据表中，除去所有的主属性，剩下的就都是**非主属性**了。
4. 查看是否存在属性对候选码的部分函数依赖或者传递函数依赖。
   1. 非主属性不存在对候选码的部分函数依赖，2NF
   2. 非主属性不存在对候选码的部分函数依赖和传递函数依赖，3NF
   3. 在第三范式的基础上，主属性存在对候选码的传递函数依赖与部分函数依赖，BCNF
5. 对于第一步：
   1. 查看所有每一单个属性，当它的值确定了，是否剩下的所有属性值都能确定。
   2. 查看所有包含有两个属性的属性组，当它的值确定了，是否剩下的所有属性值都能确定。
   3. 依次类推…

技巧：如果某属性或属性组A确定为候选码，那么包含了A的属性组，如：(A,B),(A,C),(A,B,C)等等，都不是候选码了，因为候选码的子集必须不能唯一确定一个元组，候选码中不能有候选码，这样也矛盾了，这里的(A,B),(A,C),(A,B,C)是超码

模式分解

在执行关系模式分解时，必须遵守的原则是：**保持无损连接**和**原有函数依赖关系**

7数据库设计

步骤

1. 需求分析； -->数字词典,数据结构的描述
2. 概念结构设计； ->视图 ER图
3. 逻辑结构设计； -->关系表 关系数据模型
4. 物理结构设计； -->存储安排.存储路径的建立
5. 数据库实施； -->创建数据模式,装入数据,数据库试运行
6. 数据库运行和维护。 -->性能检测,数据库重组和重构.

1 需求分析阶段

2 概念结构设计阶段

- 通过对用户需求进行综合、归纳与抽象，形成一个独立于具体DBMS的概念模型，可以用E-R图表示。这是数据库设计的关键

3 逻辑结构设计阶段

- 将概念结构转换为某个DBMS所支持的数据模型（例如关系模型），并对其进行优化（例如使用范式理论）

4 数据库物理设计阶段

- 为逻辑数据模型选取一个最适合应用环境的物理结构（包括存储结构和存取方法）

5 数据库实施阶段

- 运用DBMS提供的数据语言（例如SQL）及其宿主语言（例如C），根据逻辑设计和物理设计的结果建立数据库，编制与调试应用程序，组织数据入库，并进行试运行

6 数据库运行和维护阶段

- 数据库应用系统经过试运行后即可投入正式运行。在数据库系统运行过程中必须不断地对其进行评价、调整与修改

需求分析

数据字典

对数据库设计来讲，数据字典是进行数据收集和数据分析所获得的主要成果。数据字典是各类数据描述的集合。

数据字典通常包括数据项、数据结构、数据流、数据存储和处理过程五个部分

作用:数据字典是关于数据库中数据的描述，**在需求分析阶段建立**，是下一步进行概念设计的基础，并在数据库设计过程中不断修改、充实和完善

概念结构设计

概念模型

设计方法

- 自顶向下
- 自底向上
- 逐步扩张
- 混合策略

E-R模型

实体联系

E-R图

E-R方法是抽象和描述现实世界的有力工具

要点

- 使用长方形来表示实体型，框内写上实体名
- 椭圆型表示实体的属性，并用无向边把实体和属性连接起来。
- 用菱形表示实体间的联系，菱形框内写上联系名，用无向边把菱形分别与有关实体相连接，在无向边旁标上联系的类型，若实体之间联系也具有属性，则把属性和菱形也用无向边连接上。
- 实体间的对应关系要在表上表示出来．例如：1对多就要在菱形两边的横线上写1和n

E-R图冲突

- 属性冲突
  - （1）属性域冲突，即属性值的类型、取值范围或取值集合不同。
  - （2）属性取值单位冲突
- 命名冲突
  - （1）同名异义
  - （2）异名同义（一义多名）
- 结构冲突
  - （1）同一对象在不同应用中具有不同的抽象。例如“教材”在某一局部应用中被当作实体，而在另一局部应用中则被当作属性
  - （2）同一实体在不同局部视图中所包含的属性不完全相同，或者属性的排列次序不完全相同
  - （3）实体之间的联系在不同局部视图中呈现不同的类型。例如实体E1与E2在局部应用A中是多对多联系，而在局部应用B中是一对多联系；又如在局部应用X中E1与E2发生联系，而在局部应用Y中E1、E2、E3三者之间有联系

在初步E-R图中可能存在一些冗余的数据和实体间冗余的联系。所谓**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。**冗余数据和冗余联系容易破坏数据库的完整性，给数据库维护增加困难，应当予以消除。消除了冗余后的初步E-R图称为基本E-R图

E-R图集成

步骤

1. **合并**(消除冲突)
2. **修改和重构**(消除不必要的冗余)

转关系表

- 首先将所有的实体分别对应一个关系模式
- 1对1的实体关系，将其中一个实体的主码写到另一个实体的关系模式中．例如：A和B是1对1的关系，可以将A的主码作为普通的码写到B的关系模式中，B中不用再写A的主码
- 1对多的实体关系：将1端实体的主码作为外码写到N端实体的关系模式中并注明是外码
- 多对多的实体关系：新建一个关系模式，模式名是关系名字，主码是两侧实体的主码．不要忘记关系的属性

E-R模型转换为关系模型的规则

- 每一个实体对应一个关系模式
- 每个m：n联系对应一个关系模式

逻辑结构设计

进行关系模式设计并进行规范化处理

E-R图向关系模式的转换

1. 一个实体型转换为一个关系模式。实体的属性就是关系的属性。实体的码就是关系的码。
2. 一个m:n联系转换为一个关系模式。与该联系相连的各实体的码以及联系本身的属性均转换为关系的属性。而关系的码为各实体码的组合。
3. 一个1:n联系可以转换为一个独立的关系模式，也可以与n端对应的关系模式合并。如果转换为一个独立的关系模式，则与该联系相连的各实体的码以及联系本身的属性均转换为关系的属性，而关系的码为n端实体的码。
4. 一个1:1联系可以转换为一个独立的关系模式，也可以与任意一端对应的关系模式合并。如果转换为一个独立的关系模式，则与该联系相连的各实体的码以及联系本身的属性均转换为关系的属性，每个实体的码均是该关系的候选码。如果与某一端对应的关系模式合并，则需要在该关系模式的属性中加入另一个关系模式的码和联系本身的属性。
5. 三个或三个以上实体间的一个多元联系转换为一个关系模式。与该多元联系相连的各实体的码以及联系本身的属性均转换为关系的属性。而关系的码为各实体码的组合。
6. 具有相同码的关系模式可合并。

1:1

1. 将联系转换成一个独立的关系模式，关系模式的名称为联系的名称。联系的属性包括该联系的属性和关联两个实体的主键，联系的主键为任意两个关联实体的主键之一。
2. 将联系的属性归并到关联的两个实体的任一方，并将那一方实体集的主键加过来作为外键。

例子:一个公司有一个经理，一个经理也只能在一个公司进行任职，所以公司和经理之间是一对一的关系。公司（公司编号，公司名，地址，电话），经理（职工编号，姓名，性别，电话）。经理任职会产生任职日期属性。

转换为关系模式:

公司（公司编号，公司名，地址，电话）

经理（职工编号，姓名，性别，电话）

任职（公司编号，职工编号，任职日期） 主键为公司编号或者是职工编号

（1）公司（公司编号，公司名，地址，电话，职工编号，任职日期） 主键为公司编号，外键为职工编号
经理（职工编号，姓名，性别，电话）

（2）公司（公司编号，公司名，地址，电话）

经理（职工编号，姓名，性别，电话，公司编号，任职日期） 主键为职工编号，外键为公司编号。

1:n

将联系转换成一个独立的关系模式：名称为联系的名称，关系模式的属性两个实体的主键，联系的主键是多方（n方）实体集的主键。

将联系的属性和1端的主键加入多方（n方）

例子:一个部门中有多个员工，但是每个员工只能属于一个部门，所以部门和员工之间的关系就是一对多。部门（部门号，部门名称，电话），员工（员工号，姓名，电话）。

转换为关系模式:

1. 1方：部门（部门号，部门名称，电话）
   n方：员工（员工号，姓名，电话）
   隶属（员工号，部门号） 主键是n方
2. 部门（部门号，部门名称，电话）
   员工（员工号，姓名，电话，部门号） 主键为员工号，外键为部门号

m:n

多对多的联系只能转换成一个独立的关系模式，关系模式的名称取联系的名称，关系模式的属性取该联系所关联的所有实体集的主键和自身的属性，**关系模式的主键由所有关联的实体集共同组成。**

例子:学生（学号，姓名，性别），课程（课程号，课程名称，授课老师）。

- 转换为关系模式:

  1. 学生（学号，姓名，性别）
     课程（课程号，课程名称，授课老师）
     选课（学号，课程号，成绩） 主键为学号和课程号的组合，外键为课程号，学号

数据模型优化

- 确定数据依赖
- 对于各个关系模式之间的数据依赖进行极小化处理，消除冗余的联系。
- 按照数据依赖的理论对关系模式逐一进行分析，考查是否存在部分函数依赖、传递函数依赖、多值依赖等，确定各关系模式分别属于第几范式。
- 按照需求分析阶段得到的各种应用对数据处理的要求，分析对于这样的应用环境这些模式是否合适，确定是否要对它们进行合并或分解。
- 对关系模式进行必要的分解。

1. 数据库逻辑设计**不是唯一**的
2. 并不是规范化程度越高的关系就越优
3. 模式分解

   1. 水平分解:水平分解是把（基本）关系的元组分为若干子集合，定义每个子集合为一个子关系，以提高系统的效率。根据“**80/20原则**”.
   2. 垂直分解:垂直分解是把关系模式R的属性分解为若干子集合，形成若干子关系模式。

物理结构设计

确定数据库存储结构时要综合考虑存取时间、存储空间利用率和维护代价三方面的因素。这三个方面常常是相互矛盾的。

为了提高系统性能，数据应该根据应用情况将易变部分与稳定部分、经常存取部分和存取频率较低部分分开存放

1. 设计步骤
    1. **确定数据库的物理结构**，在关系数据库中主要指存取方法和存储结构
    2. **对物理结构进行评价**，评价的重点是时间和空间效率。
2. 设计的内容:为关系模式选择存取方法，以及设计关系、索引等数据库文件的物理存储结构。

8 DB编程

嵌入式SQL

使用游标的SQL

说明游标

```
EXEC SQL DECLARE
```

打开游标

```
EXEC SQL OPEN
```

推进游标取数据

```
EXEC SQL FETCH

INTO <主变量>[指示变量][,...]
```

关闭游标

```
EXEC SQL CLOSE
```

动态SQL

过程化SQL

存储过程与函数

理解

创建

```
Create or replace procedure 过程名
AS <过程化SQL块>
```

9关系查询优化

建索引优化(补充)

哪些情况对属性建立索引比较合适？哪些情况不适合建立索引?

适合建索引

1. 频繁作为where条件语句查询的字段
2. 关联字段需要建立索引，例如外键字段，student表中的classid，classes表中的schoolid 等
3. 排序字段可以建立索引
4. 分组字段可以建立索引，因为分组的前提是排序
5. 统计字段可以建立索引，例如count(),max()

不适合建索引

1.频繁更新的字段不适合建立索引

2.where条件中用不到的字段不适合建立索引

3.表数据可以确定比较少的不需要建索引

4.数据重复且发布比较均匀的的字段不适合建索引（唯一性太差的字段不适合建立索引），例如性别，真假值

5. 参与列计算的列不适合建索引

关系系统

- 1.支持关系数据库（关系数据结构）
  - 从用户观点看，数据库由表构成，并且只有表这一种结构。
- 2.支持选择、投影和（自然）连接运算，对这些运算不必要求定义任何物理存取路径
  - 当然并不要求关系系统的选择、投影、连接运算和关系代数的相应运算完全一样，而只要求有等价的这三种运算功能就行。

查询优化

对于给定的查询选择代价最小的操作序列，使查询过程既省时间，具有较高的效率，这就是所谓的查询优化。对于关系数据库系统，用户只要提出“做什么”，而由系统解决“怎么做”的问题。具体来说，是数据库管理系统中的查询处理程序自动实现查询优化。

关系查询优化是影响RDBMS性能的关键因素。关系系统的查询优化既是RDBMS实现的关键技术又是关系系统的优点所在。

查询优化的优点不仅在于用户不必考虑如何最好地表达查询以获得较好的效率，而且在于系统可以比用户程序的“优化”做得更好

一般准则

- 1.选择运算应尽可能先做。在优化策略中这是最重要、最基本的一条。它常常可使执行时节约几个数量级，因为选择运算一般使计算的中间结果大大变小
- 2.在执行连接前对关系适当地预处理。预处理方法主要有两种，在连接属性上建立索引和对关系排序 。
- 3.把投影运算和选择运算同时进行。如有若干投影和选择运算，并且它们都对同一个关系操作，则可以在扫描此关系的同时完成所有的这些运算以避免重复扫描关系。
- 4.把投影同其前或其后的双目运算结合起来，没有必要为了去掉某些字段而扫描一遍关系

- 5.把某些选择同在它前面要执行的笛卡尔积结合起来成为一个连接运算，连接特别是等值连接运算要比同样关系上的笛卡尔积省很多时间
- 6.找出公共子表达式。

代数优化

略(太难了

物理优化

查询树的启发式优化

见一般准则

10恢复

事务

用户定义的一个数据库操作序列，这些操作要么全做，要么全不做，是一个不可分割的工作单位

事务和程序是两个概念

- 在关系数据库中，一个事务可以是一条SQL语句，一组SQL语句或整个程序
- 一个应用程序通常包含多个事务

事务是恢复和并发控制的基本单位

事务语句

- `BEGIN TRANSACTION;`
  - 开启事务

- COMMIT;
  - 事务提交
- ROLLBACK;
  - 事务回滚

事务特性(ACID特性)

- 原子性（Atomicity）
  - 事务是数据库的逻辑工作单位
  - 事务中包括的诸操作要么都做，要么都不做
- 一致性（Consistency）
  - 事务执行的结果必须是使数据库从一个　一致性状态变到另一个一致性状态
  - 一致性状态：
    - 数据库中只包含成功事务提交的结果
  - 不一致状态：
    - 数据库中包含失败事务的结果
- 隔离性（Isolation）
  - 对并发执行而言一个事务的执行不能被其他事务干扰
  - 一个事务内部的操作及使用的数据对其他并发事务是隔离的
  - 并发执行的各个事务之间不能互相干扰
- 持续性（Durability ）
  - 持续性也称永久性（Permanence）
  - 一个事务一旦提交，它对数据库中数据的改变就应该是永久性的。
  - 接下来的其他操作或故障不应该对其执行结果有任何影响。

故障类型

- 事务故障
  - 事务内部故障可分为**预期的**和**非预期的**，其中大部分的故障都是非预期的。预期的事务内部故障是指可以通过**事务程序本身发现的事务内部故障**;非预期的事务内部故障是不能由事务程序处理的，如运算溢出故障、并发事务死锁故障、违反了某些完整性限制而导致的故障等

- 系统故障
  - 系统故障也称为**软故障**，是指数据库在运行过程中，由于硬件故障、数据库软件及操作系统的漏洞、突然停电灯情况，导致系统停止运转，所有正在运行的事务以非正常方式终止，需要系统重新启动的一类故障。这类事务不**破坏数据库**，但是**影响正在运行的所有事务。**
- 介质故障
  - 介质故障也称为**硬故障**，主要指数据库在运行过程中，由于磁头碰撞、磁盘损坏、强磁干扰、天灾人祸等情况，使得数据库中的数据部分或全部丢失的一类故障。
- 计算机病毒

恢复基本原理

冗余

> 利用存储在系统其它地方的冗余数据来重建数据库中已被破坏或不正确的那部分数据

恢复实现技术

数据转储（backup）

转储即数据库管理员定期地将整个数据库复制到磁带、磁盘或其他存储介质上保存起来的过程。这些备用的数据称为后备副本（backup）或后援副本。

- 静态转储：是在系统中**无运行事务时**进行的转储操作。
- 动态转储：是指转储期间允许对数据库进行存取或修改。即转储和用户事务可以并发执行。

方法

1. 动态海量转储
2. 动态增量转储
3. 静态海量转储
4. 静态增量转储

- 海量转存：指每次转储全部数据库
- 增量转存：指每次只转储上一次转储后更新过的数据

登记日志文件（logging）

日志文件

日志文件是用来记录事务对**数据库更新操作**的文件

登记的原则

- 严格按照时间次序
- 必须先写日志，后写数据库

（1）事务故障恢复和系统故障恢复必须用日志文件。

（2）在动态转储方式中必须建立日志文件，后备副本和日志文件结合起来才能有效地恢复数据库。

（3）在静态转储方式中也可以建立日志文件，当数据库毁坏后可重新装入后援副本把数据库恢复到转储结束时刻的正确状态，然后利用日志文件把已完成的事务进行重做处理，对故障发生时尚未完成的事务进行撤销处理。这样不必重新运行那些已完成的事务程序就可把数据库恢复到故障前某一时刻的正确状态.。

恢复策略

恢复方法
由恢复子系统利用日志文件撤消（UNDO）此事务已对数据库进行的修改
事务故障的恢复由系统自动完成，对用户是透明的，不需要用户干预

事务故障

- 反向扫描文件日志（即从最后向前扫描日志文件），查找该事务的更新操作。
- 对该事务的更新操作执行逆操作。即将日志记录中“更新前的值”写入数据库。

插入操作，“更新前的值”为空，则相当于做删除操作
删除操作，“更新后的值”为空，则相当于做插入操作
若是修改操作，则相当于用修改前值代替修改后值

（3） 继续反向扫描日志文件，查找该事务的其他更新操作，并做同样处理。
（4） 如此处理下去，直至读到此事务的开始标记，事务故障恢复就完成了。

系统故障

- Undo 故障发生时未完成的事务
- Redo 已完成的事务
  系统故障的恢复由系统在**重新启动时**自动完成，不需要用户干预

系统故障的恢复步骤：

（1）正向扫描日志文件（即从头扫描日志文件）

- 重做(REDO) 队列： 在故障发生前已经提交的事务
  这些事务既有BEGIN TRANSACTION记录，也有COMMIT记录
- 撤销 (UNDO)队列：故障发生时尚未完成的事务
  这些事务只有BEGIN TRANSACTION记录，无相应的COMMIT记录

（2） 对撤销(UNDO)队列事务进行撤销(UNDO)处理
反向扫描日志文件，对每个撤销事务的更新操作执行逆操作
即将日志记录中“更新前的值”写入数据库
（3）对重做(REDO)队列事务进行重做(REDO)处理
正向扫描日志文件，对每个重做事务重新执行登记的操作
即将日志记录中“更新后的值”写入数据库

检查点恢复技术

checkpoint

注意针对不同的事务状态采取的不同策略

DB镜像

略

11并发

多事务执行方式

(1)事务串行执行

- 每个时刻只有一个事务运行，其他事务必须等到这个事务结束以后方能运行
- 不能充分利用系统资源，发挥数据库共享资源的特点

(2)交叉并发方式 (interleaved concurrency)

- 事务的并行执行是这些并行事务的并行操作轮流交叉运行
- 是单处理机系统中的并发方式，能够减少处理机的空闲时间，提高系统的效率

(3)同时并发方式 (simultaneous concurrency)

- 多处理机系统中，每个处理机可以运行一个事务，多个处理机可以同时运行多个事务，实现多个事务真正的并行运行
- 最理想的并发方式，但受制于硬件环境
- 更复杂的并发方式机制

并发问题

- 丢失修改 (lost update)
  - 丢失修改是指事务1与事务2从数据库中读入同一数据并修改
  - 事务2的提交结果破坏了事务1提交的结果，导致事务1的修改被丢失。
- 不可重复读 (non-repeatable read)
  - 不可重复读是指事务1读取数据后，事务2执行更新操作，使事务1无法再现前一次读取结果。
- 读“脏”数据 (dirty read)
  - 事务1修改某一数据，并将其写回磁盘
  - 事务2读取同一数据后
  - 事务1由于某种原因被撤消，这时事务1已修改过的数据恢复原值
  - 事务2读到的数据就与数据库中的数据不一致，
  - 是不正确的数据，又称为“脏”数据。

并发控制机制的任务

略

- 保证事务的隔离性
- 保证数据库的一致性

封锁

- 封锁就是事务T在对某个数据对象（例如表、记录等）操作之前，先向系统发出请求，对其加锁
- 加锁后事务T就对该数据对象有了一定的控制，在事务T释放它的锁之前，其它的事务不能更新此数据对象。
- 封锁是实现并发控制的一个非常重要的技术

类型

- 排它锁（eXclusive lock，X锁）
  - 写锁
  - 若事务T对数据对象A加上X锁，则只允许T读取和修改A，其它任何事务都不能再对A加任何类型的锁，直到T释放A上的锁
- 共享锁（Share lock，S锁）
  - 读锁
  - 若事务T对数据对象A加上S锁，则其它事务只能再对A加S锁，而不能加X锁，直到T释放A上的S锁

基本锁的相容矩阵

Y相容 N不相容

X - X 不相容 X - S 不相容，S - S 相容

T_2 \ T_1	X	S	-
X	N	N	Y
S	N	Y	Y
-	Y	Y	Y

封锁协议

1级封锁协议

- 事务T在修改数据R之前必须先对其加X锁，直到事务结束才释放
- 1级封锁协议可防止丢失修改
- 在1级封锁协议中，如果是读数据，不需要加锁的，所以它不能保证可重复读和不读“脏”数据。
  - 读“脏”数据
  - 不可重复读

2级封锁协议

- 1级封锁协议+事务T在读取数据R前必须先加S锁，读完后即可释放S锁
- 2级封锁协议可以防止丢失修改和读“脏”数据。
- 在2级封锁协议中，由于读完数据后即可释放S锁，所以它不能保证可重复读。

3级封锁协议

- 1级封锁协议 + 事务T在读取数据R之前必须先对其加S锁，直到事务结束才释放

- 3级封锁协议可防止丢失修改、读脏数据和不可重复读。

活锁和死锁

活锁

饥饿．一个事务等待时间太长，似乎被锁住了，实际上可能被激活

解决办法:先来先服务的策略,类比操作系统的FCFS调度算法

死锁

简而言之就是,多个事物之间互相请求对方的资源而等待,导致永不结束

死锁的诊断与解除

1. 与操作系统类似，一般使用**超时法**或**事务等待图**法。
2. DBMS**并发控制子系统检测到死锁后**，就要设法解除。通常采用的方法是**选择一个处理死锁代价最小的事务，将其撤销，释放此事务持有的所有锁，使其他事务得以继续运行下去**

并发调度的可串行性

可串行化调度

什么样的并发调度是正确调度？**可串行化的调度**

多个事务的并发执行是正确的，当且仅当其结果与按某一次序串行地执行这些事务时的结果相同，称这种调度策略为可串行化（serializable）调度

冲突可串行化调度

两段锁协议

所有事务必须分两个阶段对数据项加锁和解锁

- 在对任何数据进行读、写操作之前，首先要申请并获得对该数据的封锁；
- 在释放一个封锁之后，事务不再申请和获得任何其他封锁。

“两段”锁的含义是事务分为两个阶段：

1. 第一阶段是获得封锁，也称为扩展阶段，在这个阶段，事务可以申请获得任何数据项上的任何类型的锁，但是不能释放任何锁
2. 第二阶段是释放封锁，也称为收缩阶段，在这个阶段，事务可以释放任何数据项上的任何类型的锁，但是不能再申请任何锁

封锁粒度

封锁的大小称为封锁的粒度

多粒度封锁

意向锁