数据库复习

第一章

数据库系统特点

1. 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

2. 数据的共享度高，冗余度底，易扩充
3. 数据独立性高

> 数据独立性包括物理独立性和逻辑独立性（本质：希望不管怎么变，想不修改应用程序）
> 物理独立性：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：指用户的应用程序与数据库的逻辑结构是相互独立的

4. 数据由数据库管理系统（DBMS）统一管理和控制

数据库结构的基础是数据模型(data model)

> 数据模型是一个描述数据结构、数据操作以及数据约束的数学形式体系(即概念及其符号表示系统)

数据模型的分层

- 概念模型：概念层次的数据模型，也称为信息模型

> 常用的概念模型有实体-联系模型(E-R模型)和面向对象模型(OO模型)

- 逻辑模型：用于描述数据库数据的逻辑结构

> 层次模型(hierarchical model)
> 网状模型(network model)
> 关系模型(relational model)
> 面向对象模型(即OO模型)
> XML模型
> 对象关系模型(object relational model)

- 物理模型：用来描述数据的物理存储结构和存取方法

数据模型的三要素

- 数据结构：描述数据库的组成对象以及对象之间的联系
- 数据操作：指对数据库中各种对象(型)的实例(值)允许执行的操作集合，包括操作及有关的操作规则
- 数据完整性约束：一组数据完整性规则，是数据、数据语义和数据联系所具有的制约和依存规则，以保证数据库中数据的正确、有效和相容

关系模型中的常用术语：

- 关系(relation)：一个关系对应一张二维表，每一个关系有一个名称，即关系名；
- 元组(tuple)：表中的一行称为一个元组；
- 属性(attribute)：表中的一列称为一个属性，每一个属性有一个名称，即属性名；

- 码(key)：也称为码键或键。表中的某个属性或属性组，它可以唯一地确定关系中的一个元组，如关系Student中的学号，它可以唯一地标识一个学生；
- 域(domain)：属性的取值范围；
- 分量(component)：元组中的一个属性值
- 外码(foreign key)：表中的某个属性或属性组，用来描述本关系中的元组(实体)与另一关系中的元组(实体)之间的联系
- 关系模式(relational schema)：通过关系名和属性名列表对关系进行描述，即二维表的表头部分(表格的描述部分)

关系模式的一般形式

> 关系名(属性名1，属性名2，...，属性名n)
> 带下画线的属性为码属性，斜体的属性为外码属性

数据库的三级模式

- **模式**(逻辑层抽象)
  - 也称为逻辑模式，对应于逻辑层数据抽象，**是数据库中全体数据的逻辑结构和特征的描述**，是所有用户的公共数据视图。
  - 模式的一个具体值称为模式的一个实例(instance)
  - 它是DBMS模式结构的中间层，既不涉及数据的物理存储细节和硬件环境，也与具体的应用程序、所使用的应用开发工具及高级程序设计语言无关
- **外模式**(视图层抽象)
  - **也称子模式或用户模式**，对应于视图层数据抽象
  - 是数据库用户能够看见和使用的局部数据的逻辑结构和特征的描述，是数据库用户的数据视图，是与某一具体应用有关的数据的逻辑表示
  - 外模式是保证数据库安全性的一个有力措施，每个用户只能看见和访问所对应的外模式中的数据，数据库中的其余数据是不可见的
- **内模式**(物理层抽象)
  - **也称存储模式，对应于物理层数据抽象**，它是数据的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

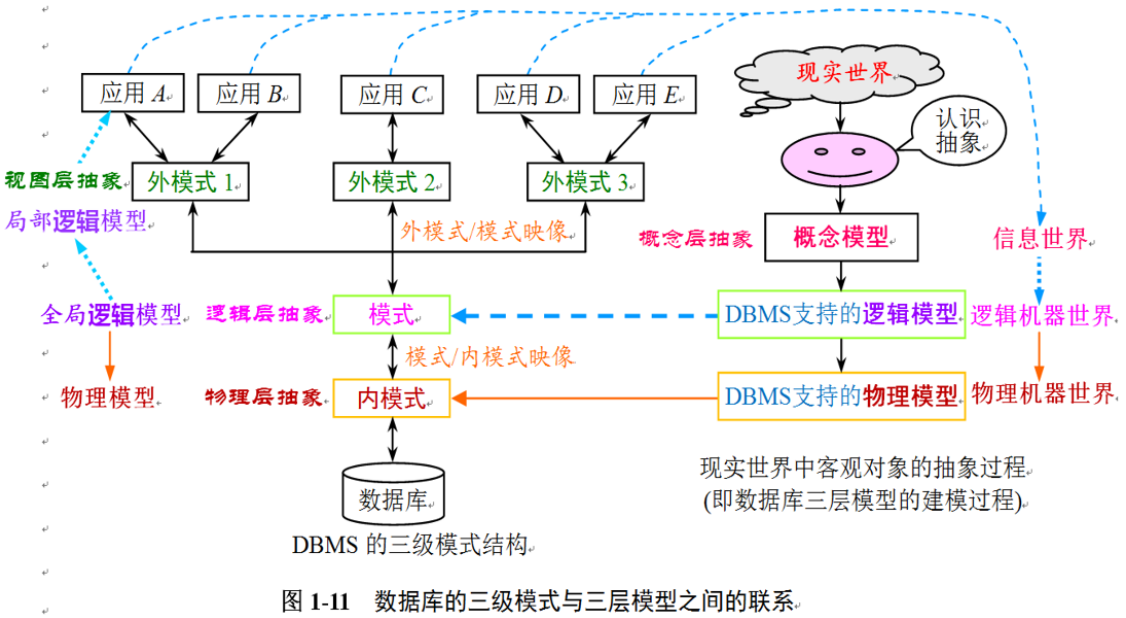


图 1-11　数据库的三级模式与三层模型之间的联系

数据库的两层映像功能与数据独立性

- 外模式/模式映像

> 对应于一个模式可以有多个外模式。对于每一个外模式，数据库管理系统都有一个模式/外模式映像，它定义了该外模式与模式之间的对应关系
> 在各自的外模式描述中定义外模式/模式映像
> 保证了数据与应用程序的逻辑独立性，简称为数据的逻辑独立性

- 模式/内模式映像

> 数据库中只有一个模式，也只有一个内模式，模式/内模式映像是唯一的，它定义了数据全局逻辑结构与存储结构之间的对应关系
> 在模式描述中定义模式/内模式映像
> 保证了数据与应用程序的物理独立性，简称为数据的物理独立性

DBMS的组成

> 查询处理器：对用户请求的SQL操作进行查询优化，从而找到一个最优的执行策略，然后向存储管理器发出命令，使其执行。
> 存储管理器：根据执行策略，从数据库中获取相应的数据,或更新数据库中相应的数据。
> 事务管理器：负责保证系统的完整性，保证多个同时运行的事务不发生冲突操作，以及保证当系统发生故障时数据不会丢失

数据库系统(database system, DBS)，是指在计算机系统中引入数据库后的系统，

一般由**数据库、数据库管理系统(及其应用开发工具)、应用系统、数据库管理员和最终用户**构成

例题

数据库管理系统能实现对数据库中数据的查询、插入、修改和删除，这类功能称为（ ）。

A 数据定义功能

B 数据管理功能

C 数据操纵功能

D 数据控制功能

区分不同实体的依据是（ ） 。

A 名称

B 属性

C 对象

D 概念

数据的逻辑独立性是指（ ）。

A 内模式改变，模式不变

B 模式改变，内模式不变

C 模式改变，外模式和应用程序不变

D 内模式改变，外模式和应用程序不变

数据库系统的数据独立性体现在（ ） 。

A 不会因为数据的变化而影响到应用程序

B 不会因为数据存储结构与数据逻辑结构的变化而影响应用程序

C 不会因为存储策略的变化而影响存储结构

D 不会因为某些存储结构的变化而影响其他的存储结构

要保证数据库的数据独立性,需要修改的是( ) .

选择一项:

- A. 模式与外模式
- B. 模式与内模式
- C. 三层之间的两种映射
- D. 三层模式

要保证数据库的逻辑数据独立性,需要修改的是( ) .

选择一项:

- A. 模式与外模式的映射
- B. 模式与内模式之间的映射
- C. 模式
- D. 三层模式

第二章

关系数据结构

- 关系

> 关系模型的数据结构非常简单，它就是二维表，亦称为关系
> 关系模型中，现实世界的实体以及实体间的各种联系都是用关系来表示

- 域
- 笛卡尔积
- 空值(null)
- 关系模式

> 关系的描述称为关系模式(relation schema)
> 形式化地表示为：`r(U, D, DOM, F)`
>
> > r 为关系名
> > U 为组成该关系的属性名的集合
> > D 为属性集U中所有属性所来自的域的集合
> > DOM 为属性向域的映像集合，
> > F 为属性间数据的依赖关系集合(即体现各属性取值之间的“关联”性)
> > **一般简写位r(U)**

- 码

> **超码**：对于关系r的一个或多个属性的集合A，如果属性集A可以唯一地标识关系r中的一个元组，则称属性集A为关系r的一个超码 (superkey) 。
> **候选码**：如果属性集A是关系r的超码，且属性集A的任意真子集都不能成为关系r的超码，则称属性集A为候选码 (candidate key) 。
> **主码**：若一个关系有多个候选码，则可以选定其中的一个候选码作为该关系的主码

关系完整性约束

- **实体完整性规则**

  > - 若属性集A是关系r的主码，则A不能取空值null
  > - 如果主码是由若干个属性的集合构成，则要求构成主码的每一个属性的值都不能取空值

- **参照完整性规则**

  > - 若属性(或属性集)F是关系r的外码，它与关系s的主码Ks相对应，则对于关系r中的每一个元组在属性F上的取值要么为空值null，要么等于关系s中某个元组的主码值
  > 翻译成人话就是外码要么与对应的关系主码一致,要么为空

- **用户自定义完整性**

  > - 任何关系数据库管理系统都应该支持实体完整性和参照完整性
  > - 用户定义完整性是针对某一具体应用要求来定义的约束条件，它反映某一具体应用所涉及的数据必须满足的语义要求
  > 人话就是: 要满足业务规定的约束, 如唯一性

关系操作

- 关系操作的特点是集合操作方式，即操作的对象和结果都是集合。这种操作方式也称为一次一个集合的方式。相应地，非关系数据模型的数据操作方式则为一次一个记录的方式
- 关系模型中的关系操作有**查询操作**和**更新操作(插入、删除和修改)**两大类
- 查询操作是关系操作中最主要的部分。查询操作又可以分为选择(select)、投影(project)、连接(join)、除(divide)、并(union)、交(intersection)、差(except)、笛卡尔积等

关系操作可用两种方式来表示

- 关系代数是用代数方式表达的关系查询语言。
- 关系演算是用逻辑方式表达的关系查询语言

> 均是抽象的查询语言，在表达能力上是完全等价的

关系代数运算:

- 并运算

  > 基本形式为: $r \cup s$ (结果为二者元组之和去除重复行)

- 交运算

  > 基本形式为: $r \cap s$ (结果为二者重复行)

- 差运算

  > 基本形式为: $r - s$ (前者去除二者重复行)

- 笛卡尔积

  > 基本形式为: $r \times s$
  >
  > 结果列数为二者属性列数之和,行数为二者元素数乘积
  >
  > 即关系 r 中的每一个元组去和关系 s 中的每一个元组拼接

- 选择

> 基本形式为: $\sigma_{条件}(r)$, 其作用是将符合条件的元组过滤出来单独构成一个新的关系
>
> 如 : 对于关系 $r(Sno, Sname)$ 有 $\sigma_{Sno='20222005017'}(r)$

- 投影

> 基本形式为: $\prod_{投影的列}(r)$, 其作用是之将几个属性单独列出来作为一个新的关系
>
> 可以类比是 `SELECT SNO, SNAME ...` 的作用
>
> 如对于关系 $r(Sno, Sname)$ 有 $\prod_{Sno}(r)$

- 连接

> 基本形式为 $r \bowtie_{\theta} s$
>
> θ 连接运算就是从关系 r 和 s 的笛卡尔积 $r \times s$ 中，选取 r 关系在 A 属性集上的值与 s 关系在 B 属性集上的值满足连接谓词 θ 的所有元组
>
> 即 : $r \bowtie_{\theta} s = \sigma_{\theta}(r \times s)$
>
> 连接运算中有两种最常用、最重要的连接，一种是等值连接(equijoin)，另一种是自然连接(natural join)。
>
> θ 为等值比较谓词的连接运算称为等值连接。
> **自然连接是一种特殊的等值连接**，它要求两个参与连接的关系具有公共的属性集，并在这个公共属性集上进行等值连接；同时，还要求将连接结果中的重复属性列去除掉，即在公共属性集中的列只保留一次

- 象集

> **本质上是一次选择行的运算和一次选择列的运算。**
> **求x1在表A中的象集，就是先选出所有x属性中x=x1的那些行，然后选择出不包含x1的那些列**
>
> 总结就是取先进行一次选择,在进行一次投影

- 除法

> 关系R和关系S拥有共同的属性B、C , $R \div S$ 得到的属性值就是关系R包含而关系S不包含的属性，即A属性
>
> 讲解：有关系 R 和关系 S
> 关系 R，包含 A、B、C 三个属性
>
A	B	C
a1	b1	c2
a2	b3	c7
a3	b4	c6
a1	b2	c1
a4	b6	c6
a2	b2	c3
a1	b2	c3
>
> 关系 S，包含 B、C、D 三个属性
>
B	C	D
b1	c2	d1
b2	c1	d1
b2	c3	d2
>
> 在R关系中A属性的值可以取{ a1，a2，a3，a4 }
>
> a1值对应的象集为 { (b1,c2) , (b2,c1) , (b2,c3) }
>
> a2值对应的象集为 { (b3,c7) , (b2,c3) }

> a3值对应的象集为{ (b4,c6) }
>
> a4值对应的象集为{ (b6,c6) }
>
> 关系S在B、C上的投影为{ (b1,c2) , (b2,c1) , (b2,c3) }
>
> 只有a1值对应的象集包含关系S的投影集，所以只有a1应该包含在A属性中
>
> 所以 $R \div S$ 为

A
a1

例题

五种基本关系代数运算是( ).

选择一项:

- A. ∪ - × σ π
- B. ∪ - σ π
- C. ∪ ∩ × σ π
- D. ∪ ∩ σ π

正确答案是：∪ - × σ π

关系代数运算是以( )为基础的运算 .

选择一项:

- A. 关系运算
- B. 谓词演算
- C. 集合运算
- D. 代数运算

正确答案是：集合运算

第三章

SQL特点

- 综合统一

  > 集数据定义语言DDL、数据操纵语言DML、数据控制语言DCL的功能于一体

- 高度非过程化：描述做什么，不涉及怎么做。
- 面向集合的操作方式

  > 采用集合操作方式，其操作对象、操作结果都是元组的集合

- 同一种语法结构提供两种使用方式

  > SQL语言既是自含式语言，又是嵌入式语言。在两种不同的使用方式下，其语法结构基本上是一致的

- 语言简洁，易学易用

  > SQL语言的动词非常少，主要包括：
  >
  > > 数据查询： SELECT;
  > >
  > > 数据更新： INSERT、UPDATE、DELETE;
  > > 数据定义： CREATE、DROP、ALTER;
  > >
  > > 数据控制： GRANT、REVOKE

查询

```
/* 基本格式如下：
SELECT 字段列表
FROM 表名列表
WHERE 条件列表
GROUP BY 分组字段列表
HAVING 分组后条件列表
ORDER BY 排序字段列表
LIMIT 分页参数 ;
*/
```

> 查询指定字段

```
# 查询指定字段 SELECT 字段1, 字段2 ... FROM 表名;
# 查询全部字段 SELECT * FROM 表名;
# 去除重复记录 SELECT DISTINCT 字段列表 FROM 表名; (即去重数据)
# 可以给字段名起别名 SELECT 字段1 AS 别名 FROM 表名;
SELECT name, workno, age FROM emp;
SELECT * from emp;
SELECT DISTINCT workaddress FROM emp;
SELECT workaddress as '工作地址' FROM emp;
```

> 条件查询

```
# SELECT 字段列表 FROM 表名 WHERE 条件列表
```

符号	意义
> , >= , < , <= , = , !=	与数学意义一致
AND , && , OR , || , NOT , !	逻辑运算符
[NOT]BETWEEN...AND...	在某个范围之内（含最大最小）
[NOT]IN(....)	在 in 之后列表中的值， 多选一

符号	意义
[NOT] LIKE 占位符	模糊匹配（_ 匹配单一字符， % 匹配任意个字符）
IS [NOT] NULL	是NULL

```
# 查询年龄为18，20，40的员工信息
SELECT * FROM emp WHERE age = 18 || age = 20 || age = 40;
SELECT * FROM emp WHERE age in (18, 20, 40);
# 查询姓名为两个字的员工信息
SELECT * FROM emp WHERE name LIKE '__';
# 查询身份证号以X结尾的员工信息
SELECT * FROM emp WHERE idcard LIKE '%X';
```

> 聚合函数

```
# SELECT 聚合函数(字段列表) FROM 表名;
# 所有的NULL值不参与计算
```

函数	功能
count	统计数量
max	最大值
min	最小值
avg	平均值
sum	求和

演示

```
SELECT count(id) as '总数' FROM emp;
SELECT avg(age), max(age), min(age) FROM emp;
SELECT count(id) FROM emp WHERE workaddress = '西安';
```

> 分组查询

```
# SELECT 字段列表 FROM 表名 [WHERE 条件] GROUP BY 字段分组名 [HAVING 分组后的过滤条件];
```

WHERE 是分组之前进行过滤， HAVING 是分组之后进行过滤

WHERE 不能对聚合函数进行判断， HAVING 可以

```
# 根据性别分组 ，统计男性员工和女性员工的数量
SELECT gender, count(*) from emp GROUP BY gender;
# 根据性别分组 ，统计男性员工和女性员工的平均年龄
SELECT gender, avg(age) FROM emp GROUP BY gender;
# 查询年龄小于45的员工，并根据工作地址分组，获取员工数量大于等于3的工作地址
SELECT workaddress, count(*) FROM emp WHERE age < 45 GROUP BY workaddress HAVING
count(*) > 3;
```

> 排序查询

```
#  SELECT 字段列表 FROM 表名 ORDER BY 字段1 排序方法1， 字段2， 排序方法2....;
```

如果第一个字段值一样，才会根据第二个字段排序

> ASC 升序（默认）
>
> DESC 降序

```
SELECT name, age, entrydata FROM emp ORDER BY age , entrydata desc ;
```

> 分页查询

```
SELECT 字段列表 FROM 表名 LIMIT 起始索引, 查询记录数;
```

```
SELECT name, age FROM emp LIMIT 0, 10;
SELECT  name age FROM emp LIMIT 10, 10;
```

连接

分类：

- 连接查询
  - 内连接： 相当于查询 A， B 交集的部分
  - 外连接
    - 左外连接： 查询左表的全部数据，以及两张表的交集数据
    - 右外连接：查询右表的全部数据，以及两张表的交集数据
  - 自连接

> 内连接

隐式内连接

```
SELECT 字段列表 FROM 表一, 表二 WHERE 条件;
```

显示内连接

```
SELECT 字段列表 FROM 表一 [INNER] JOIN 表二 ON 连接条件
```

PS： 可以结合别名， 方便使用

> 外连接

左外连接

```
SELECT 字段列表 FROM 表一 LEFT [OUTER] JOIN 表二 ON 条件;
```

右外连接

```
SELECT 字段列表 FROM 表一 RIGHT [OUTER] JOIN 表二 ON 条件;
```

> 自连接

```
SELECT 字段列表 FROM 表A 别名A JOIN 表二 别名二 ON 条件;
```

可以将自己看成两张表

> 联合查询

吧多次查询的结果联合起来

```
SELECT * from 表A
UNION [ALL]
SELECT * FROM 表B;
```

PS: 联合查询的基本条件为两个表查询出来列数， 内容要一样

PS： 不加ALL 的话会去重， 加了ALL不会去重

> 子查询

在SQL语句里面嵌套SQL语句

PS： 理解为用SQL语句的返回的表作为临时表进行多表查询

分类：

- 标量子查询（子查询结果为单个值）
- 列子查询 （子查询结果为一列）
- 行子查询（子查询结果为一行）
- 表子查询（子查询结果为多行多列）

__> 标量子查询（子查询结果为单个值）

```
SELECT * from 表一 WHERE 字段 > (SELECT ID FROM 表二 WHERE 条件);
```

__> 列子查询 （子查询结果为一列）

常见操作符

操作符	描述
IN	在指定内容里面
NOT IN	不在指定内容里面
ANY	有一个满足即可
SOME	与ANY一致
ALL	子查询的结果都必须满足

```
SELECT 字段 FROM 表一 WHERE 字段 > (SELECT ID FROM 表二);
```

__> 行子查询（子查询结果为一行）

```
SELECT * FROM 表一 WHERE (字段一, 字段二...) = (SELECT 字段一, 字段二... from 表二
WHERE 条件);
```

__> 表子查询（子查询结果为多行多列）

常用操作符为 IN

```
SELECT * FROM 表一 WHERE (字段一, 字段二...) IN (SELECT 字段一, 字段二... from 表二
WHERE 条件);
```

第四章

数据库设计过程

- 需求分析

  > 了解和分析系统将要提供的功能及未来数据库用户的数据需求

- 概念设计

  > 根据需求分析中得到的信息，运用适当的工具将这些需求转化为数据库的概念模型

- 逻辑设计

  > 将数据库的概念模型转化为所选择的数据库管理系统支持的逻辑数据模型，即数据库模式

- 模式求精

  > 分析并发现数据库逻辑模式存在的问题，并进行改进和优化, 减少数据冗余，消除更新、插入与删除异常

- 物理设计

  > 虑数据库要支持的负载和应用需求，为逻辑数据库选取一个最适合现实应用的物理结构

- 应用与安全设计

E-R模型三要素: 实体, 属性, 联系

属性分类: 简单属性和复合属性 / 单值属性和多值属性 / 派生属性 / 空值(NULL)
实体集——矩形
属性——椭圆
多值属性——双椭圆
派生属性——虚线椭圆
属性与实体之间——连线

参与联系集的实体集的数目称为联系集的度

E-R模型的约束主要有：

- 映射约束 （重点）

  > 联系的种类有1:1(一对一), 1:m(一对多), m:n(多对多)

- 码约束 （重点）

  > 当一实体集存在多个候选码时，主码选择原则：
  >
  > - 选择属性长度最短的候选码；
  > - 选择包含单个属性的码，而不是复合候选码；
  > - 选择在数据库系统生命周期内属性值最少变化的候选码；
  > - 选择在数据库系统生命周期内更可能包含唯一值的候选码

- 依赖约束(难点)

  > **依赖约束是指联系中一种实体的存在依赖于该联系集中联系或其他实体集中实体的存在**
  > 联系中一种实体的存在依赖于该联系集中联系的存在，称为实体集与联系集之间的依赖约束，并将依赖于联系集而存在的实体集称为**依赖实体集**；
  > 联系中一种实体的存在依赖于其他实体集中实体的存在，称为实体集之间的依赖约束，并将依赖于其他实体集而存在的实体集称为**弱实体集**

- 参与约束

在现实世界中存在一类实体集，其属性不足以形成主码，它们必须依赖于其它实体集的存在而存在，我们称这样的实体集为弱实体集 (weak entity set)。与此相对应，其属性可以形成主码的实体集称为强实体集

弱实体集所依赖的强实体集称为标识实体集 (identifying entity set)。弱实体集必须与一标识实体集相关联才有意义，该联系集称为标识联系集 (identifying relationship set)

销货单实体集的存在是依赖于商品销售联系集的存在，也就是说，没有商品销售联系，就没有销货单实体，即销货单实体集与商品销售联系集之间存在依赖约束，销货单是依赖实体集

属性冲突、命名冲突、结构冲突

E-R模型设计原则
忠实性、简单性、避免冗余

> **忠实性**
> 设计应忠实于应用需求，这是首要的也是最重要的原则。即实体集、属性、联系集都应当反映现实世界及根据所了解的现实世界去建模。
> **简单性**
> 除非有绝对需要，否则不要在设计中增加更多成分；
> 只需要对数据库使用者所关心、感兴趣的属性建模 。
> **避免冗余**
> 原则：一个对象只存放在一个地方。

选择实体集还是属性?

选择实体集还是联系集？（依赖约束、多值联系的建模）

多元联系转化为二元联系：联系实体集、依赖实体集或弱实体集

例题

第五章

函数依赖

平凡函数依赖： 在Y函数依赖于X函数的基础上，即X → Y时，如果Y ⊂ X 那么称X→Y是平凡的函数依赖，

如果有Y ⊂ X，那么X → Y 一定成立

原因：因为Y ⊂ X，那么Y必然是X中的一部分，因为X确定了，那么自然其子集也确定了，整体可以决定部分

非平凡函数依赖： 当关系中属性集合Y不是属性集合X的子集时，存在函数依赖X→Y，则称这种函数依赖为非平凡函数依赖。

[数据库 函数依赖及范式（最通俗易懂） - 蔡军帅 - 博客园 (cnblogs.com)]

基于函数依赖理论，关系模式可分成

- 第一范式(1NF)

  > 第一范式(1NF)：属性不可分割，即每个属性都是不可分割的原子项。(实体的属性即表中的列)

- 第二范式(2NF)

  > 第二范式(2NF)：满足第一范式；且不存在部分依赖，即非主属性必须完全依赖于主属性。(主属性即主键；完全依赖是针对于联合主键的情况，非主键列不能只依赖于主键的一部分)

- 第三范式(3NF)

> 第三范式(3NF)：满足第二范式；且不存在传递依赖，即非主属性不能与非主属性之间有依赖关系，非主属性必须直接依赖于主属性，不能间接依赖主属性。（A -> B, B ->C, A -> C）

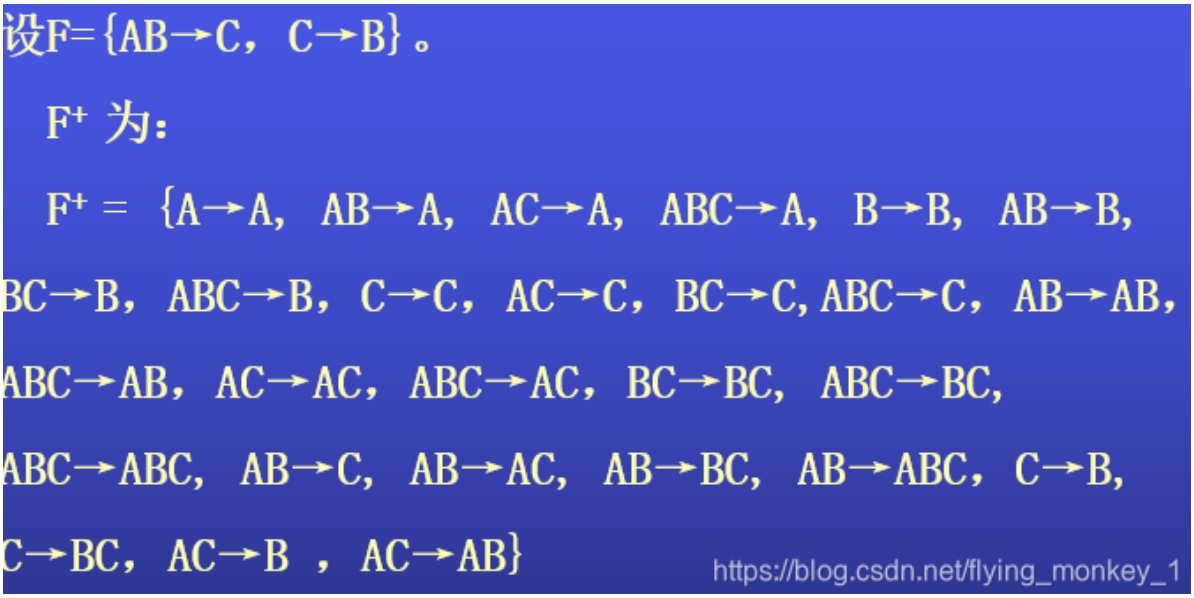
- Boyce-Codd范式(BCNF)

> BCNF要求所有的非平凡函数依赖 $\alpha \rightarrow \beta$ 中的 α 是超码，而3NF则放松了该约束，允许α不是超码。
> 若关系模式属于BCNF范式就一定属于3NF范式。反之则不一定成立

这几种范式的要求一个比一个严格

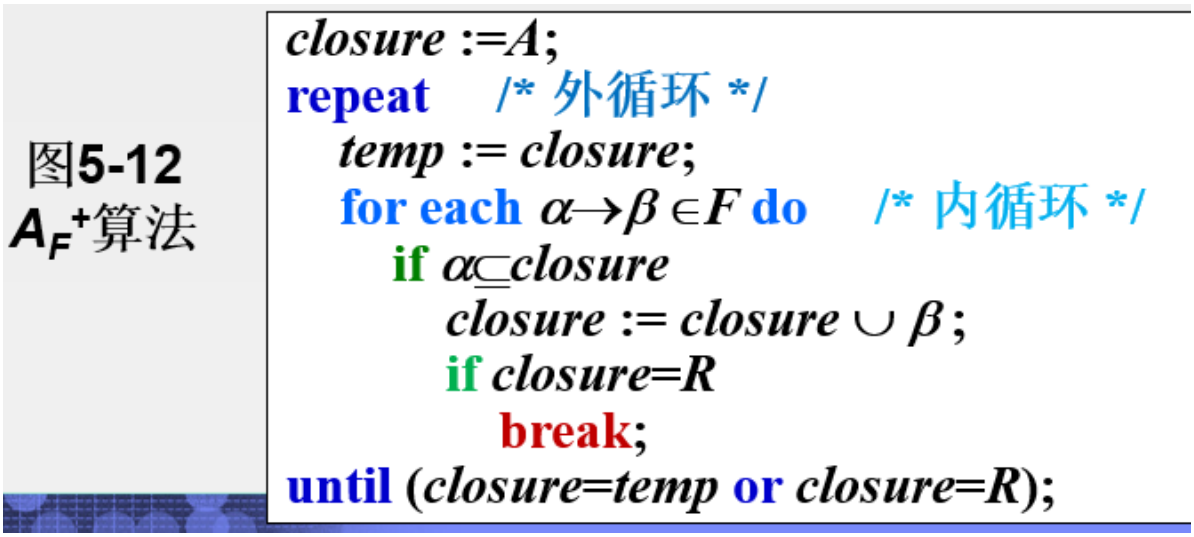
函数依赖集闭包
若给定函数依赖集F，可以证明其他函数依赖也成立，则称这些函数依赖被F逻辑蕴涵
令F为一函数依赖集，F逻辑蕴涵的所有函数依赖组成的集合称为F的闭包，记为 F^+

F^+ 就是函数依赖集闭包



令r(R)为关系模式，F为函数依赖集，A 是 R 的子集，则称在函数依赖集F下由A函数确定的所有属性的集合为函数依赖集F下属性集A的闭包，记为 A^+

ps：可以理解为 A^+表示所有 A 可以决定的属性



计算属性集闭包的作用可归纳如下：

- 验证$\alpha\to\beta$是否在F^+中：看是否有$\beta\subseteq\alpha_F{}^+$。
- 判断α是否为$r(R)$的超码：计算α^+，看其是否包含R的所有属性。如$(AG)^+=ABCGHI$，则AG为$r(R)$的超码。
- 判断α是否为$r(R)$的候选码：若α是超码，可检验α包含的所有子集的闭包是否包含R的所有属性。若不存在任何这样的属性子集，则α是$r(R)$的候选码。
- 计算F^+。对于$\forall\ \gamma\subseteq R$，可通过找出$\gamma^+$；
  则有：对$\forall\ S\subseteq\gamma^+$，可输出一个 $\gamma\to S$。

■ 定义5.19　给定关系模式$r(R)$及函数依赖集F，则将$r(R)$分解成$r_1(R_1)$、$r_2(R_2)$的分解是无损连接分解，

➡ 当且仅当F^+包含函数依赖$R_1\cap R_2\to R_1$或$R_1\cap R_2\to R_2$.

■ 即：在F下，$R_1\subseteq(R_1\cap R_2)^+$或$R_2\subseteq(R_1\cap R_2)^+$。

■ 因此，当一个关系模式分解为两个关系模式时，该分解为无损连接分解的充要条件是两分解关系的公共属性包含$r_1(R_1)$的码 或 $r_2(R_2)$的码。

➡ 即：$R_1\cap R_2$ 是关系 $r_1(R_1)$ 或 $r_2(R_2)$ 的超码。

在关系规范式中,分解关系的基本原则是().

I.实现无损连接

II.分解后的关系相互独立

III.保持原有的依赖关系

选择一项:

- A. Ⅰ和Ⅱ
- B. Ⅰ和Ⅲ
- C. Ⅰ
- D. Ⅱ

正确答案是: Ⅰ和Ⅱ

如果A->B ,那么属性A和属性B的联系可能是( ) .

选择一项:

- A. 一对多
- B. 多对一
- C. 多对多
- D. 以上都不是

正确答案是: 多对一

设有关系模式R(S,D,M),其函数依赖集:F = {S→D,D→M},则关系模式R的规范化程度最高达到( ).

选择一项:

- A. 1NF
- B. 2NF
- C. 3NF
- D. BCNF

正确答案是: 2NF

第七章

数据库创建语句:

```
CREATE DATABASE <数据库名称>
ON
([PRIMARY][Name=<逻辑文件名>,]
FILENAME='<物理文件名>'
[,SIZE=<大小>]
[,MAXSIZE=<可增长的最大大小>]
[,FILEGROWTH=<增长比例>])
LOG ON
([Name=<逻辑文件名>,]
FILENAME='<物理文件名>'
[,SIZE=<大小>]
[,MAXSIZE=<可增长的最大大小>]
[,FILEGROWTH=<增长比例>])
```

ON:指定显示定义用来存储数据库部分的磁盘文件(数据文件).
PRIMARY:该选项是一个关键字，指定主文件组中的文件。
LOG ON:指明事务日志文件的明确定义。
NAME：指定数据库的逻辑名称，它是在SQL Server系统中使用的名称，是数据库在SQL Server中的标识符.
FILENAME: 指定数据库文件名和存储路径.
SIZE:指定数据库的初始容量大小.
MAXSIZE:指定文件可增长到的最大值。如果没有指定，则文件可以不断增长直到充满磁盘.
FILEGREOWTH:指定文件每次增加容量的大小，当指定数据为“0“时,表示文件不增长

```
CREATE DATABASE School
ON PRIMARY
    (Name=School_data,
    FILENAME='D:\temp\School_data.mdf',
    SIZE=5MB,
    MAXSIZE=20MB,
    FILEGROWTH=10%)
LOG ON
(Name=School_log,
FILENAME="D:\temp\School_log.ldf",
SIZE=3MB,
MAXSIZE=10MB,
FILEGROWTH=1MB)
```

数据库修改语句

```
ALTER DATABASE <databaseName>
  {  ADD FILE {<filespec> [, ...n]} [TO FILEGROUP  <filegroupName>]
      | ADD LOG FILE {<filespec> [, ... n]}
      | REMOVE FILE <logicalFileName>
      | ADD FILEGROUP <filegroupName>
      | REMOVE FILEGROUP <filegroupName>
      | MODIFY FILE <filespec>
      | MODIFY FILEGROUP <filegroupName> <filegroupProperty>
  }
```

删除数据库

```
DROP DATABASE <databaseName>
```

数据库表的定义

```
 CREATE TABLE <tableName>
      (  <columnName1> <dataType>
                [DEFAULT <defaultValue>] [null | NOT null],
         [ <columnName2> <dataType>
                [DEFAULT <defaultValue>] [null | NOT null],  … ]
        [ [CONSTRAINT <constraintName1>] {UNIQUE | PRIMARY KEY}
                (<columnName> [, <columnName>…]) [ON <filegroupName>], ... ]
        [ [CONSTRAINT <constraintName2>]
                FOREIGN KEY (<columnName1> [, <columnName2>…] )
                REFERENCE [<dbName>.owner.]<refTable>
                       (<refColumn1> [, <refColumn2>… ]) [ON <filegroupName>],
... ]
     ) [ON <filegroupName>]
```

> ：定义约束的名字，属于可选项

实际使用创建主键外键:

```
CREATE TABLE cartoon(
    cartoon_id VARCHAR(7) UNIQUE PRIMARY KEY NOT NULL COMMENT "番剧的唯一标识",
    cartoon_name VARCHAR(30) COMMENT "番剧的名字",
    cartoon_cover VARCHAR(50) COMMENT "番剧封面的存储路径",
    cartoon_permit INT COMMENT "是否为VIP, VIP为1, 普通为0"
) COMMENT "番剧基本信息表";

# carousel( cartoon_id[外键], carousel_cover )
CREATE TABLE carousel(
    cartoon_id VARCHAR(7) UNIQUE NOT NULL COMMENT "关联cartton表的标识",
    carousel_cover VARCHAR(50) COMMENT "走马灯展示的图片",
    FOREIGN KEY (cartoon_id) REFERENCES cartoon (cartoon_id)
) COMMENT "走马灯展示信息";
```

修改表

```
ALTER TABLE <tableName> ADD <columnName> <dataType>
ALTER TABLE <tableName> ADD CONSTRAINT <constraintName>
ALTER TABLE <tableName> DROP <constraintName>
ALTER TABLE <tableName> ALTER COLUMN <columnName> <newDataType>
```

基本表在修改过程中，不可以删除列

删除表

```
DROP TABLE TempTable
```

索引的分类:

聚集或非聚集，非聚集索引就是普通索引，一个基本表可以建立多个普通索引。

- 每个基本表仅能建立一个聚集索引

- 聚集索引按搜索码值的某种顺序(升/降序)来重新组织基本表中的记录
- 即索引的顺序就是基本表记录存放的顺序
- 聚集索引可以极大地提高查询速度，但是给数据的修改带来困难
- 建立了聚集索引的基本表一般仅执行查询操作，很少进行更新操作，这在数据仓库中使用得较多

```
CREATE [UNIQUE] [CLUSTERED | NONCLUSTERED] INDEX <indexName>
 ON <tableName> ( <columnName1> [ASC | DESC]
                                [, <columnName2> [ASC | DESC] … ] )
 [ON <filegroupName>]
```

> CLUSTERED | NONCLUSTERED： 表示建立聚集或非聚集索引，默认为非聚集索引
>
> ：索引的名称，索引是数据库中的对象，因此索引名在一个数据库中必须唯一；
>
> ([ASC | DESC] [, [ASC | DESC] ...])：指出为哪个表的哪些属性建立索引
>
> [ASC | DESC]为按升序还是降序建立索引，默认为升序

删除索引:

```
DROP INDEX <indexName> ON <tableName>
```

DML - 对数据增删改

```
/*添加数据*/
# 给指定字段添加数据 INSERT INTO 表名(字段名1， 字段名2，...) VALUES(值1， 值2...);
INSERT INTO name(id) VALUES(100);
# 给全部字段添加数据(要字段一一对应) INSERT INTO 表名 VALUES(值1， 值2...);
INSERT INTO name VALUES(123, "yyym");
/*批量添加数据*/
# 指定字段 INSERT INTO 表名(字段名1， 字段名2，...) VALUES(值1， 值2...),(值1， 值
2...)...;
# 全部字段 INSERT INTO 表名 VALUES(值1， 值2...), (值1， 值2...)...;
INSERT INTO name(id) VALUES(456),(789),(741);
INSERT INTO name VALUES(569, "lr580"), (563, "Hura"), (4743, "cgc");
```

```
/*修改数据*/
# UPDATA 表名 SET 字段名1 = 值1, 字段名2 = 值2... [WHERE 条件];
# 没有条件则说明修改所有值
UPDATE name SET name = "cst7" WHERE id = 789;
```

```
/*删除数据*/
# DELETE FROM 表名 [WHERE 条件];
# 没有条件则删除所有的数据
DELETE FROM name WHERE id = 741;
```

视图的概念

视图是一个虚拟表，其内容由查询定义。同真实的表一样，视图包含一系列带有名称的列和行数据。但是，数据库中只存放了视图的定义，而并没有存放视图中的数据，这些数据存放在原来的表中。使用视图查询数据时，数据库系统会从原来的表中取出对应的数据。因此，视图中的数据是依赖于原来的表中的数据的。一旦表中的数据发生改变，显示在视图中的数据也会发生改变。同样对视图的更新，会影响到原来表的数据。

视图是存储在数据库中的查询的SQL语句，它主要出于两种原因：安全原因，视图可以隐藏一些数据，例如，员工信息表，可以用视图只显示姓名、工龄、地址，而不显示社会保险号和工资数等；另一个原因是可使复杂的查询易于理解和使用。这个视图就像一个“窗口”，从中只能看到你想看的数据列。这意味着你可以在这个视图上使用SELECT *，而你看到的将是你在视图定义里给出的那些数据列。

视图的作用

视图是在原有表或者视图的基础上重新定义的虚拟表，这可以从原有的表上选取对用户有用的信息，忽略次要信息，其作用类似于筛选。视图的作用归纳为如下几点：

1. 使操作简单化

> 视图需要达到的目的就是所见即所需。视图不仅可以简化用户对数据的理解，也可以简化他们的操作。那些被经常使用的查询可以被定义为视图，从而使得用户不必为以后的操作每次指定全部的条件。

2. 增加数据的安全性

> 通过视图，用户只能查询和修改指定的数据。指定数据之外的信息，用户根本接触不到。这样可以防止敏感信息被未授权的用户查看，增强机密信息的安全性。

3. 提高表的逻辑独立性

> 视图可以屏蔽原有表结构变化带来的影响。例如原有表增加列和删除未被引用的列，对视图不会造成影响。同样，如果修改表中的某些列，可以使用修改视图来解决这些列带来的影响。

创建视图

```
CREATE VIEW view_student1(stu_id,stu_name,stu_class) AS SELECT id,name,class
FROM student;
CREATE OR REPLACE VIEW view_student1(stu_id,stu_name,stu_class) AS SELECT
id,name,class FROM student;
```

修改视图

```
ALTER VIEW view_student1 AS SELECT id,name,class FROM student where id  in
(select id from student );
```

删除视图

```
DROP VIEW IF EXISTS view_student;
```

查看视图定义

```
describe view_student;
```

什么是“游标（Cursor）”

游标是SQL 的一种数据访问机制 ，游标是一种处理数据的方法。众所周知，使用SQL的select查询操作返回的结果是一个包含一行或者是多行的数据集，如果我们要对查询的结果再进行查询，比如（查看结果的第一行、下一行、最后一行、前十行等等操作）简单的通过select语句是无法完成的，因为这时候索要查询的结果不是数据表，而是已经查询出来的结果集。游标就是针对这种情况而出现的。

我们可以将“ 游标 ”简单的看成是结果集的一个指针，可以根据需要在结果集上面来回滚动，浏览我需要的数据。

二、游标的操作——五步走
声明游标—>打开游标—>读取数据—>关闭游标—>删除游标

```
declare productcursor cursor scroll       --第一步：声明游标
for
select * from productinfo where vendname='上海华测'
GO
open productcursor                        --第二步：打开游标
Go
--读取数据开始                            --第三步：获取数据
fetch next from productcursor --读取当前行的下一行，并使其置为当前行(刚开始时游标置于表头的前一行，即若表是从0开始的，游标最初置于-1处，所以第一次读取的是头一行)
fetch prior from productcursor --读取当前行的前一行，并使其置为当前行
fetch first from productcursor --读取游标的第一行，并使其置为当前行（不能用于只进游标）
fetch last from productcursor  --读取游标的最后一行，并使其置为当前行（不能用于只进游标）
fetch absolute 2 from productcursor --读取从游标头开始向后的第2行，并将读取的行作为新的行
fetch relative 3 from productcursor --读取从当前行开始向后的第3行，并将读取的行作为新的行
fetch relative-2 from productcursor --读取当前行的上两行，并将读取的行作为新的行
--读取数据结束
GO
close productcursor                       --第四步：关闭游标
Go
deallocate productcursor                  --第五步：删除游标
Go
```

第八章

查询处理的过程：

（1）基本步骤：①语法分析与翻译；②查询优化；③查询执行。

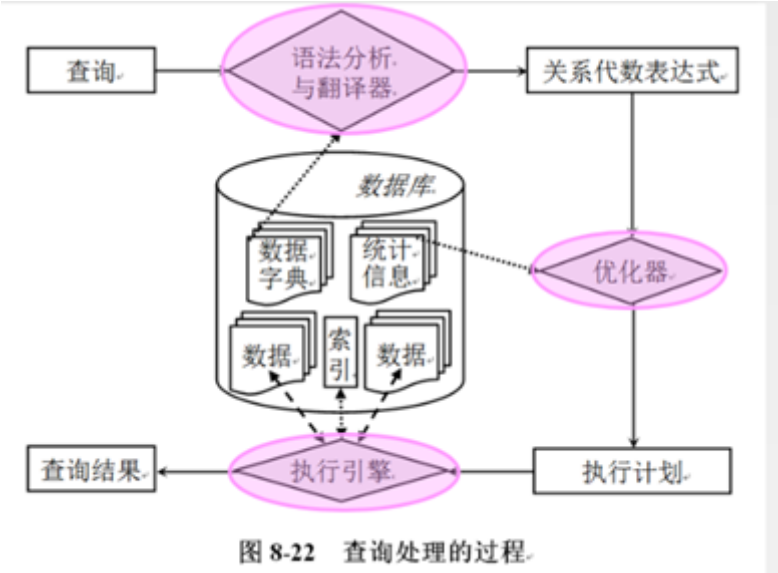


图 8-22　查询处理的过程

（2）语法分析与翻译器：

①检查用户查询地语法，利用数据字典验证查询中出现的关系名和属性名等是否正确；

②构造该查询语句地语法分析树表示，并将其翻译成关系代数表达式。

（3）查询执行计划与查询优化器：

执行一个查询，不仅需要提供关系代数表达式，还要对该关系代数表达式加上注释来说明如何执行每个关系运算。生产查询执行计划。

不同地查询执行计划会有不同的代价。构造具有最小查询执行代价的查询执行计划称为查询优化，由查询优化器来完成。

查询优化是影响RDBMS性能的关键因素。

（4）查询执行引擎

查询执行引擎根据输入的查询执行计划，调用相关算法实现查询计算，并将计算结果返回给用户。

有效地对内存缓冲区进行管理是影响查询执行性能的非常重要的方面。

查询代价的度量：

磁盘存取时间（最重要的代价）、执行一个查询所用CPU时间以及在并行/分布式数据库系统中的通信开销

查询优化的过程：

①逻辑优化，即产生逻辑上与给定关系代数表达式等价的表达式；

②代价估计，即估计每个执行计划的代价；

③物理优化，即对所产生的表达式以不同方式做注释，产生不同的查询执行计划。

逻辑优化和物理优化的概念：

逻辑优化：系统尝试找出一个与给出的查询表达式等价，但执行效率更高的查询表达式；

物理优化：为处理查询选择一个详细的策略。

启发式优化规则：

尽早执行选择操作；

尽早执行投影运算；

索引的设计：

常见的存储方式：索引方法、聚集方法和Hash方法。目前使用最普遍的是B+树索引。

特点：

（1）存取路径和数据是分离的；

（2）存取路径可以由用户建立和删除，也可以由系统动态建立和删除；

（3）存取路径的物理组织可以采用顺序文件、B+树文件或Hash文件。