```yaml
title: 大学课程 | 数据库基础
tags:
  - 数据库
  - 大学课程
categories:
  - 学习笔记
abbrlink: 41347
reward: true
copyright: true
date: 2019-07-10 20:15:15
cover: https://npm.elemecdn.com/justlovesmile-img/bgok2.jpg
top_img: https://npm.elemecdn.com/justlovesmile-img/bgok2.jpg
```

作者博客：Justlovesmile's BLOG

大二数据库课程笔记

第一章 绪论

1.1数据库系统概述

- 应用数据库的主要目的是为了：**共享数据。**

1.1.1 数据库的4个基本概念

1.四个基本概念：**数据**（data），**数据库**（DB），**数据库管理系统**（DBMS）和**数据库系统**（DBS）

- 数据：
  - 定义：描述事物的**符号记录**称为数据
  - 数据是数据库中存储的基本对象
  - 数据的含义称为数据的语义，数据与其语义是不可分的
  - 记录是计算机表示和存储数据的一种格式或方法
- 数据库：
  - 定义：数据库是**长期存储**在计算机内，**有组织的**，**可共享**的**大量数据**的集合，数据库中的数据按一定的数据模型**组织**，**描述**和**存储**，具有**较小的冗余度**，**较高的数据独立性**和**易扩展性**，并可为各种**用户共享。**
- 数据库管理系统：
  - 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
  - 数据库管理系统和操作系统一样是计算机的基础软件
  - 主要功能：
    （1）数据定义功能：
    数据定义语言（DDL）
    （2）数据组织，存储和管理
    提高存储空间利用率和方便存取
    （3）数据操纵功能
    数据操纵语言（DML）
    （4）数据库的事务管理和运行管理
    1. 保证安全性完整性，多用户对数据的并发使用
    2，发生故障后的系统恢复
    （5）数据库的建立和维护功能

    （6）其他功能

- 数据库系统：
  - 定义：数据库系统是由**数据库**，**数据库管理系统**（及其应用开发工具），**应用程序**和**数据库管理员**（DBA）组成的**存储**，**管理**，**处理和维护数据**的系统。

1.1.2数据管理技术的产生和发展

- 数据管理是指对数据进行分类，组织，编码，存储，检索和维护
- 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
- 数据库管理的三个阶段

	人工管理阶段	文件系统阶段	数据库系统极端
硬件背景	无直接存取存储设备	磁盘，磁鼓	大容量磁盘，磁盘阵列
软件背景	没有操作系统	有文件系统	有数据库管理系统
处理方式	批处理	联机实时处理	联机实时处理，分布处理，批处理
数据的共享程度	无共享，冗余度较大	共享性差，冗余度大	共享性高，冗余度小
数据的独立性	不独立，完全依赖于程序	独立性差	具有高度的物理独立性和一定的逻辑独立性
数据的结构化	无结构	记录内有结构，整体无结构	整体结构化，用数据模型描述
数据控制能力	应用程序自己控制	应用程序自己控制	由数据库管理系统提供数据安全性，完整性，并发控制和恢复能力

1.1.3 数据库系统的特点

- 数据**结构化**：数据库系统实现整体数据的结构化，这是数据库的主要特征之一，也是数据库系统与文件系统的本质区别。数据之间具有联系
- 数据的**共享性高**，**冗余度低**且**易扩充**
- 数据独立性高：**物理独立性**（物理存储），**逻辑独立性**（逻辑结构）
- 数据由数据库管理系统统一管理和控制
  （1）数据的安全性保护：防止不合法使用造成的数据泄密和破坏
  （2）数据的完整性保护：正确性，有效性和相容性
  （3）并发控制
  （4）数据库的恢复

综上所述：

> *数据库*是**长期存储**在计算机内的**有组织，大量，共享**的数据集合。它可以供各种用户**共享**，具有**最小冗余度**和**较高的数据独立性**。数据库管理系统在数据库建立，运算和维护时对数据库进行统一控制，以保证数据的完整性和安全性，并在多用户同时使用数据库时进行并发控制，在发生故障后对数据库进行恢复。

1.2 数据模型

- 数据模型：是对现实世界数据特征的抽象
- 数据模型是数据库系统的核心和基础

1.2.1 两类数据模型

- **概念模型**：也叫信息模型，是按**用户观点**来对数据和信息建模，主要用于数据库设计
- **逻辑模型和物理模型**：
  逻辑模型主要包括层次模型，网状模型，关系模型，面向对象模型和对象关系数据模型，半结构化数据模型等。
  物理模型是对数据最底层的抽象，描述数据在系统内部的表示方式和存取方法，或在磁盘或磁带上的存储方式和存取方法，是面向**计算机系统**的。按**计算机观点**。

1.2.2 概念模型

---

1. 信息世界中的基本概念
   （1）实体：客观存在并可相互区别的事物。
   （2）属性：实体所具有的某一特性。
   （3）码：唯一标识实体的属性集。
   （4）域：属性的取值范围
   （5）实体性：用实体名及其属性名集合来抽象和刻画同类实体
   （6）实体集：同一类型实体的集合
   （7）联系：分为实体内联系和实体间联系。实体间联系有1对1，1对多，多对多。

- 不同实体是根据**属性值**去区分的。

2. 概念模型的一种表示方法：实体-联系方法（E-R图/E-R模型）

1.2.3 数据模型的组成要素

- 数据模型通常由**数据结构，数据操作**和**数据的完整性约束条件**三部分组成。

1. 数据结构：描述数据库的组成对象以及对象之间的联系。是对系统的**静态特性**的描述。
2. 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，包括操作及有关的操作规则。是对系统的**动态特性**的描述。
3. 数据的完整性约束条件：是一组完整性规则。包括：实体完整性，参照完整性，自定义完整性约束

1.2.4 常用的数据模型

- 层次模型
- 网状模型
- 关系模型
- 面向对象数据模型
- 对象关系数据模型
- 半结构化数据模型

1.2.5 层次模型

---

1. 层次模型的数据结构
   （1）有且只有一个结点没有双亲结点，这个结点称为根结点
   （2）根以外的其他结点有且只有一个双亲结点
   - 层次模型像一颗倒立的树，结点的双亲是唯一的。
2. 层次结构的数据操纵和完整性约束条件
3. 层次结构的优缺点：
   优点：
   （1）层次模型的数据结构比较简单清晰
   （2）层次数据库的查询效率高
   （3）提供了良好的完整性支持
   缺点：
   （1）显示生活中很多联系是非层次性的
   （2）可能会引入冗余数据

（3）查询子女结合点必须经过双亲结点
（4）由于结构严密，层次命令趋于程序化

1.2.6 网状模型

- 网状数据模型的典型代表是DBTG系统，亦称CODASYL系统

1. 网状模型的数据结构：
（1）允许一个以上的结点无双亲
（2）一个结点可以有多于一个的双亲

- 层次模型中子女结点与双亲结点的联系是唯一的而在网状模型中这种联系可以不唯一。
2.网状模型的数据操纵与完整性约束

3. 网状模型的优缺点：
优点：
（1）直接描述现实世界
（2）良好的性能，存取效率较高
缺点：
（1）结构比较复杂
（2）网状模型的DDL，DML复杂，不易掌握
（3）记录之间的联系是通存取路径实现的，加重用户编写负担

1.2.7 关系模型

---

1. 关系模型的数据结构：
术语：

- 关系：通常来说对应一张表
- 元组：表中的一行
- 属性：
- 码：唯一确定一个元组
- 域：属性的取值范围
- 分量：元组的一个属性值
- 关系模式：关系模式要求关系是规范化的，最基本的一条是每一个分量是一个不可分的数据项。

2. 关系模型的数据操纵与完整性约束条件：

- 数据操纵包括：查询，插入，删除和更新数据（集合操作，操作对象和操作结果都是关系）
- 完整性约束条件包括三大类：**实体完整性，参照完整性和用户定义的完整性。**
3.关系模型的优缺点：
优点：
（1）严格的数学概念
（2）概念单一
（3）存取路径对用户透明
缺点：
（1）存取路径对用户隐蔽
（2）查询效率不如格式化数据模型

1.3 数据库系统的结构

1.3.1 数据库系统模式的概念

- 在数据模型中有“型”和“值”的概念：型是指对某一类数据的结构和属性的说明，值是型的一个具体赋值。
- 模式（schema）是数据库中全体数据的逻辑结构和特征的描述。它仅仅涉及型的描述，不涉及具体的值
- 模式相对稳定，实例相对变动

1.3.2 数据库系统的三级模式结构

- 数据的三级模式：**外模式，模式，内模式**

1. 外模式：也称为子模式（subschema）或用户模式，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**，是与某一应用有关的数据的逻辑表示。
2. 模式：模式也称为逻辑模式，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。**
3. 内模式：内模式也称为存储模式，一个数据库只有一个内模式，它是数据**物理结构和存储方式**的描述，是数据库内部的组织方式。

1.3.3 数据库的二级映像功能和数据独立性

---

1. 外模式/模式映像：

- 模式描述的是数据的全局逻辑结构，外模式描述的是数据的局部逻辑结构。
- 同一个模式可以有任意多个外模式。
- 当模式改变时，由数据库管理员对各个外模式/模式映像作相应的改变，可以使外模式保持不变。
- 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。

2. 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，所以模式/内模式映像是唯一的

- 当数据库的存储结构改变时，由数据库管理员对模式/内模式映像作相应改变，可以使模式保持不变，从而应用程序也不必改变。保证了数据与程序的物理独立性，简称数据的物理独立性

> 在数据库的三级模式结构中，数据库模式即全局逻辑结构是数据库的中心和关键

1.4 数据库系统的组成

1.硬件平台及数据库：
要求：
（1）足够大的内存，存放操作系统，DBMS的核心模块，数据缓冲区，应用程序
（2）足够大磁盘等设备存方数据库，以及备份
（3）较高通道能力，提高数据传输率
2，软件：
（1）DBMS
（2）os
（3）与数据库接口的高级语言及其编译系统
（4）以DBMS为核心的应用开发工具
（5）为特定应用环境开发的数据库应用系统
3，人员

- 数据库管理员：
  （1）决定数据库中的信息内容和结构
  （2）决定数据库的存储结构和存取策略
  （3）定义数据的安全性要求和完整性约束条件
  （4）监控数据库的使用和运行
  （5）数据库的改进和重组，重构
- 系统分析员：
- 数据库设计人员：
- 应用程序员：
- 最终用户：
  （1）偶然用户（2）简单用户（3）复杂用户

第二章 关系数据库

2.1 关系数据结构及形式化定义

2.1.1 关系

---

1. 域：一组具有相同数据类型的值的集合
2. 笛卡儿积：域上的一种集合运算
3. 关系：R（D1，D2 ... Dn）
4. 候选码：某一属性组的值能唯一标识一个元组，而其子集不能
5. 主码，全码
6. 关系可以有三种类型：**基本关系（基本表），查询表，视图表**

- 基本关系的性质：
  （1）列是同质的，每一列来自同一域
  （2）不同的列可出自同一域
  （3）列的顺序无所谓
  （4）任意两个元组的候选码不能取相同的值
  （5）行的顺序无所谓
  （6）分量必须取原子值，即**每一个分量都必须是不可分的数据项**

2.1.2 关系模式

- 关系模式：关系的描述 R（U，D，DOM，F）
  R是关系名，U为组成该关系的属性名的集合，D为U中属性所来自的域，DOM为属性向域的映像集合，F为属性间数据的依赖关系

2.1.3 关系数据库

- 关系数据库的型也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，通常就称为关系数据库

2.1.4 关系模式的存储结构

- 在关系数据库中实体及实体间的联系都用表来表示
- 表是关系数据库的逻辑模型
- 在关系数据库的物理组织中，有的物理数据组织由操作系统完成，有的自己申请文件，进行存储管理

2.2 关系操作

2.2.1 基本的关系操作

- 常用操作：**查询，插入，删除，修改**
- 查询又包括：选择，投影，连接，除，并，差，交，笛卡儿积

2.2.2 关系数据语言的分类

- **关系代数** 和 **关系演算**
- 介于两者之间的结构化查询语言SQL（高度非过程化的语言）

2.3 关系的完整性

- 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。

2.3.1 实体完整性

- 主属性不能为空

2.3.2 参照完整性

- 若属性F是基本关系R的外码，他与基本关系S的住吗相对应，则对于R中的每一个元组在F上的值：
  （1）要么，全为空
  （2）要么，都与S中的元组的主码值相对应

- 外码：若F是R中的一个或一组属性，但不是关系R的码，K是S的主码，如果F与K相对应，则称F是R的外码，并称基本关系R是参照关系，基本关系S为被参照关系（R与S可为同一个）

- 外键并不一定要与相应的主键同名

2.4 关系代数

- 关系代数是一种抽象的查询语言，它用对关系的运算来表达查询

2.4.1 传统的集合运算

1. 并（union）
2. 差（except）
3. 交（intersection）
4. 笛卡儿积（cartesian product）

- 条件：①目相同（属性数相同）②相对应的属性来自同一域

2.4.2 专门的关系运算

- 作为关系数据系统，最小应具备的关系运算是 **选择、投影、连接**
- 数据库中五种基本运算：交，并，投影，选择，笛卡儿积

1. 选择σ

2. 投影Π

3. 连接▷◁
   （1）等值连接
   （2）非等值连接
   （3）自然连接：两个关系中进行比较的分量必须是**同名的属性组**，并且在结果中把**重复的属性列去掉。**
   - 悬浮元组：被舍弃的元组
   - 外连接：连接结果不仅包含符合连接条件的行同时也包含自身不符合条件的行。包括左外连接、右外连接和全外连接。
   - 自动去重

4. 除运算（÷）

2.5 元组演算*

第三章 关系数据库标准语言SQL

3.1 SQL概述

3.1.1 SQL的产生和发展

- 不同软件厂商对SQL的基本命令集进行了不同程度的扩充和修改

3.1.2 SQL的特点

1. 综合统一：SQL集**数据定义语言，数据操纵语言，数据控制语言**的功能于一体。
2. 高度非过程化
3. 面向集合的操作方式
4. 以同一种语言结构提供多种使用方式（既独立又可嵌入）
5. 功能简洁，易学易用

3.1.3 SQL的基本概念

- 外模式包括若干视图和部分基本表，模式包括若干基本表，内模式包括若干存储文件。
- 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，一个表可以带若干个索引，索引也可以放在存储文件中
- **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隐蔽
- 视图是从一个或多个基本表导出的虚表。它本身不独立存在数据库中，即**数据库中只存放视图的定义**而不存放视图的数据。
- 视图建立后，在**数据字典**中存放的是 **产生视图的表定义**
- 在SQL中，视图是由**基本表或视图**产生的虚表。

3.2 学生-课程数据库实例

```SQL
create database S_T;
use S_T;
create table Student
(Sno char(9) primary key,
Sname char(20) unique,
Ssex char(2),
Sage smallint,
Sdept char(20)
);
create table Course
(Cno char(4) primary key,
Cname char(40) not null,
Cpno char(4),
Ccredit smallint,
foreign key(Cpno) references Course(Cno)
);
create table SC
(Sno char(9),
Cno char(4),
Grade smallint,
primary key (Sno,Cno),
foreign key (Sno) references Student(Sno),
foreign key (Cno) references Course(Cno)
);
```

3.3 数据定义

3.3.1 模式的定义与删除

1. 定义模式

```SQL
create schema <模式名> authorization <用户名>;
```

若没指定模式名，隐含为用户名
在模式定义后可以紧接表定义

```SQL
create schema test authorization xiemingjie
create table tab1
(no char(9) primary key,
name char(20) unique,
sex char(2),
age smallint,
);
```

2. 删除模式

```SQL
drop schema <模式名> <CASCADE | RESTRICT>;
```

CASCADE为级联，表示在删除时同时把该模式下所有数据对象全部删除
RESTRICT为限制，表示只有没有任何向下属的对象时才执行删除操作

3.3.2 基本表的定义，删除与修改

- 数据类型

数据类型	含义
char(n),character(n)	长度为n的定长字符串
varchar(n),charactervarying(n)	最大长度为n的变长字符串
clob	字符串大对象
blob	二进制大对象
int，integer	长整数（4字节）

数据类型	含义
smallint	短整数（2字节）
bigint	大整数（8字节）
numeric（p，d）	定点数，由p位数字（不包括符号，小数点）组成，小数点后面有d位数字
decimal（p，d），dec（p，d）	同numeric
double precision	取决于机器精度的双精度浮点数
real	取决于机器精度的单精度浮点数
float（n）	可选精度的浮点数，精度至少为n位数字
boolean	逻辑布尔量
date	日期，格式YYYY-MM-DD
time	时间，格式 HH：MM：SS
timestamp	时间戳类型
interval	时间间隔类型

1. 定义基本表

```SQL
create table <表名>
(col1 类型 条件
col2 .........
);
```

2. 修改基本表

```SQL
ALTER TABLE<表名>
[ADD [COLUMN]<新列名><数据类型>[完整性约束]]
[ADD <表级完整性约束>]
[DROP [COLUMN]<列名>[ CASCADE | RESTRICT ]]
[DROP CONSTRAINT<完整性约束名>[ RESTRICT | CASCADE ]]
[ALTER COLUMN<列名><数据类型>];
```

3. 删除基本表

```SQL
DROP  TABLE<表名>[RESTRICE|CASCADE];
```

3.3.3 索引的建立与删除

- 建立索引：加快查询速度的有效手段
- 类型：顺序文件索引，B+树索引，散列索引，位图索引

1. 建立索引：

```SQL
CREATE [UNIQUE] [CLUSTER] INDEX <索引名>
ON <表名> (<列名>[<次序>][,<列名>[<次序>]]…) ;
```

次序有：ASC（升序），DESC（降序）
unique指此索引的每一个索引值只对应唯一的数据记录
cluster 建立的索引是聚簇索引

2. 修改索引

```SQL
ALTER INDEX  <旧索引名> RENAMA TO <新索引名>;
```

3. 删除索引

```SQL
DROP INDEX <索引名>;
```

- 删除索引时，系统会同时从数据字典中删去有关该索引的描述
- **数据字典：记录了数据库中所有的定义信息**

3.4 数据查询

3.4.1 单表查询

```SQL
SELECT  [ ALL | DISTINCT ] <目标列表达式>[,<目标列表达式>]…
FROM <表名或视图名> [ (<SELECT语句>) [AS]<别名> ]
[WHERE <条件表达式>]
```

```
[GROUP BY <列名1>[ HAVING <条件表达式> ] ]
[ORDER BY <列名2>[ ASC|DESC ] ];
```

- 用户可以指定别名来改变查询结果的列标题
- select不仅可以是表中属性列，也可以是表达式
- 在select语句中使用group by Sno 时， Sno 必须出现在**select** 子句中

1.where子句 常用的查询条件

查询条件	谓词
比较	=,>,<,>=,<=,!=,<>,!>,!<; not+上述比较运算符
确定范围	between（下限）and（上限）， not between and
确定集合	in， not in
字符匹配	like， not like（通配符%:任意长度，_：单个字符）（若本身含有通配符需要： ESCAPE转义）
空值	is null， is not null
多重条件（逻辑运算）	and，or，not

SQL

```
select Cno,Ccredit
from Course
where Cname like 'DB\_Design' ESCAPE '\' ;
```

- ESCAPE“： 表示“\”为转码字符
- 注：%可代表任意长度的字符串，例：a%b（aghjgb，avb，ab...）
  _可代表任意单个字符，例：a_b (acb，aob...) 2.选择若干元组
- 消除取消重复的行：

SQL

```
SELECT DISTINCT Sno
FROM SC;
```

3.order by子句

SQL

```
select Sno,Grade
from SC
where Cno='3'
order by Grade DESC;
```

- 对于空值，排列时的次序由具体系统实现决定

4.聚集函数

SQL

```
COUNT (*)                                  /* 统计元组个数*/
COUNT ([DISTINCT|ALL]<列名>)     /*统计一列中值的个数*/
SUM ([DISTINCT|ALL]<列名>)         /*计算一列值的总和*/
AVG   ([DISTINCT|ALL]<列名>)        /*计算一列值的平均值*/
MAX   ([DISTINCT|ALL]<列名>)       /*求一列值中的最大值*/
MIN     ([DISTINCT|ALL]<列名>)      /*求一列值中的最小值*/
```

注：where 子句中是不能用聚集函数作为条件表达式

但可以用having：

SQL

```
SELECT Sno, AVG (Grade)
FROM SC
GROUP   BY   Sno
HAVING AVG (Grade) >=90;
```

- **聚集函数只能用在select子句和group by 中的having子句**
- 当聚集函数遇到空值时，除了count（*）外，都跳过空值只处理非空值

3.4.2 连接查询

1. 等值与非等值连接查询：
   连接查询的where 子句用来连接两个表的条件称为 连接条件 或 连接谓词 连接谓词中的列名 称为 连接字段

- 一般格式：[<表名1>.]<列名1><比较运算符>[<表名2>.]<列名2>

2. 自身连接：一个表与自己连接，取别名
3. 外连接：

- 连接结果不仅包含符合连接条件的行同时也包含自身不符合条件的行。包括左外连接、右外连接和全外连接。
  `select Student.Sno,Sname,Ssex,Sage,Cno,Grade from Student left outer join SC on (Student.Sno=SC.Sno)`

4. 多表连接
   `select Student,Sno,Sname,Cname,Grade from Student,SC,Course where Student.Sno=SC.Sno and SC.Cno = Course.Cno`

3.4.3 嵌套查询

- 查询块：一个select-from-where 语句

1. 带IN谓词的子查询：

```SQL
SELECT Sname
FROM Student
WHERE Sno IN
                    (SELECT Sno
                      FROM  SC
                      WHERE Cno='2' ) ;
```

- 父查询，子查询
  -子查询的select语句中不能使用order by 子句（只能最终查询结果使用）
  -不相关子查询：子查询条件不依赖于父查询
- 相关子查询：子查询条件依赖于父查询

2. 带有比较运算符的子查询：
3. 带有any（some）或all谓词的子查询

```SQL
select Sname
from Student
where Sage<ALL
(select Sage
from Student
where Sdept='CS'
);
```

4. 带有exists谓词的子查询：exists谓词的子查询不返回任何数据，只产生逻辑真值或逻辑假值。

```SQL
select Sname
from Student
where exists
(select *
from sc
where Sno=Student.Sno and Cno='1'
);
```

3.4.4 集合查询

- 并UNION
- 交INTERSECT
- 差EXCEPT

Select。。。
From。。。
Where。。。
Union
Select。。。
From。。。
where。。。;

3.4.5 基于派生表的查询

- 派生表：出现在from子句中的子查询

```SQL
select Sno,Cno
from SC,(select Sno,Avg(Grade) from SC group by Sno) as Avg_sc(avg_sno,avg_grade)
where SC.Sno = Avg_sc.avg_sno and SC.Grade > Avg_sc.avg_grade
```

3.5 数据更新

3.5.1 插入数据

1. 插入
   （1）插入元组：

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

（2）插入子查询结果：

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

2. 修改数据（更新）

```SQL
UPDATE<表名>
SET<列名>=<表达式>[,<列名>=<表达式>]…
[WHERE<条件>];
```

3. 删除数据

```SQL
DELETE
FROM <表名>
[WHERE <条件>];
```

3.6 空值的处理

1. 空值的产生
2. 空值的判断（is null， is not null）
3. 空值的约束条件（定义中有**not null**或**unique**或**码属性**不能为空）
4. 空值的算术运算，比较运算和逻辑运算

3.7 视图

3.7.1 定义视图

1. 建立视图：

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

- WITH CHECK OPTION；加上这句话后，以后对该视图进行插入，修改，删除操作时，自动加上满足视图定义的条件。
- **行列子集视图**：从单个基本表导出的，并且只去掉某些行和某些列，但保留了主码的视图
- 带表达式的视图：带虚拟列，如2019-Sage
- 分组视图：带有聚集函数和group by 子句

2. 删除视图
`DROP VIEW<视图名>[CASCADE];`

3.7.2 查询视图

- 与基本表的查询相似
- 视图消解：对视图查询时进行有效性检查，若存在，则从数据字典中取出视图的定义，把定义中的子查询和用户的查询结合起来，转换成对基本表的查询，然后再执行修正了的查询

3.7.3 更新视图

- 视图是不实际存储数据的虚表
- 并不是所有的视图都是可更新的，一般的，行列子集视图是可更新的
- DB2规定：
  (1)若视图是由两个以上基本表导出的,则此视图不允许更新
  (2)若视图的字段来自字段表达式或常数,则不允许对此视图执行 INSERT和 UPDATE操作,但允许执行 DELETE操作。
  (3)若视图的字段来自聚集函数,则此视图不允许更新。
  (4)若视图定义中含有 GROUP BY子句,则此视图不允许更新。
  (5)若视图定义中含有 DISTINCT短语,则此视图不允许更新
  (6)若视图定义中有嵌套查询,并且内层查询的FROM子句中涉及的表也是导出该视图的基本表,则此视图不允许更新。

3.7.4 视图的作用

1. 视图能简化用户的操作
2. 视图使用户能以多个角度看待统一数据
3. 视图对重构数据库提供了一定程度的逻辑独立性
4. 视图能够对机密数据提供安全保护
5. 适当利用视图可以更清晰地表达查询

第四章 数据库安全性

4.1 数据库安全性概述

- 数据库的安全性是指保护数据库以防止不合法使用所造成的数据泄露，更改或破坏

4.1.1 数据库的不安全因素

1. 非授权用户对数据库的恶意存取和破坏
   措施：（1）用户身份鉴别（2）存取控制（3）视图
2. 数据库中重要或敏感的数据被泄露
   措施：（1）强制存取控制（2）数据加密存储（3）加密传输（4）审计日志
3. 安全环境的脆弱性

4.1.2 安全标准简介

- TCSEC和CC
- 四个方面描述安全性级别：安全策略，责任，保证，文档

4.2 数据库安全性控制

4.2.1 用户身份鉴别

1. 静态口令鉴别
2. 动态口令鉴别
3. 生物特征鉴别
4. 智能卡鉴别

4.2.2 存取控制

- 存取控制机制包括**定义用户权限**和**合法权限检查**两部分
  （1）定义用户权限：
  权限：用户对数据对象的操作权力称为权限，用户权限定义后经过编译存储再数据字典中，被称为安全规则或授权规则
  （2）合法权限检查：
  每当用户发出存取数据库的操作请求后(请求一般应包括操作类型、操作对象和操作用户等信息)数据库管理系统查找数据字典,根据安全规则进行合法权限检查,若用户的操作请求超出了定义的权限,系统将拒绝执行此操作。
  定义用户权限和合法权限检查机制一起组成了数据库管理系统的存取控制子系统
- 在**自主存取控制**方法中,用户对于不同的数据库对象有不同的存取权限,不同的用户对同一对象也有不同的权限,而且用户还可将其拥有的存取权限转授给其他用户。因此自主存取控制非常灵活。
- 在**强制存取控制**方法中,每一个数据库对象被标以一定的密级,每一个用户也被授予某一个级别的许可证。对于任意一个对象,只有具有合法许可证的用户才可以存取。强制存取控制因此相对比较严格。

4.2.3 自主存取控制方法

- 用户权限是由两个要素组成的：**数据库对象**和**操作类型**
- 在数据库系统内，定义存取权限称为**授权**
- 在关系数据库中，存取控制的对象不仅有数据本身，还有数据库模式
- 关系数据库系统中的**存取权限**：

（1）对于数据库模式：

对象	操作类型
模式	create schema
基本表	create table，alter table
视图	create view
索引	create index

（2）对于数据：

对象	操作类型
基本表和视图	select，insert，update，delete，references，all privileges
属性列	select，insert，update，references，all privileges

4.2.4 授权：授予与收回

- 在SQL中使用grant 和revoke语句向用户授予或收回对数据的操作权限

1. GRANT

- 一般格式为：

```SQL
GRANT  <权限>[, <权限>]...
on <对象类型> <对象名>[,<对象类型><对象名>]...
to <用户> [, <用户>]...
[with grant option];
```

```SQL
grant select
on table Student
to user1;
```

将指定操作对象的指定操作权限授予指定的用户。
如果指定了with grant option子句，则获取某种权限的用户还可以把这种权限再授予其他的用户，没有则只能使用该权限。

2. REVOKE

- 一般格式：

```SQL
REVOKE <权限> [,<权限>]...
on <对象类型><对象名>[,<对象类型><对象名>]...
from <用户>[,<用户>]...[cascade|restrict];
```

```SQL
revoke select
on table Student
from public;     /*收回所有用户对Student表的查询权限*/
```

- 用户可以“自主”地决定将数据的存取权限授予何人，决定是否也将“授权”的权限授予他人。因而称这样的存取控制是自主存取控制。

3. 创建数据库模式的权限

- 一般格式：

```SQL
create user <username> [ with ] [ DBA | RESOURCE | CONNECT ]
```

- 只有系统的超级用户才有权建立一个新的数据库用户
- 新创建的数据库用户有三种权限：connect，resource，dba
- 若没有指明权限，则默认该用户connect权限，不能创建新用户，不能创建模式，基本表，只能登陆数据库。
- 拥有resource权限的用户能创建基本表和视图，并成为其属主，不能创建模式与新用户，属主可以使用grant语句
- 拥有DBA权限的用户是系统中的超级用户

4.2.5 数据库角色

- 数据库角色是被命名的一组与数据库操作相关的的权限，角色是权限的集合。

1. 角色创建

```SQL
create role <角色名>
```

2. 给角色授权

```SQL
grant <权限>[,<权限>]...
on <对象类型> 对象名
to <角色> [,<角色>]
```

3. 将一个角色授予其他的角色或用户

```SQL
grant <角色1>[,<角色2>]
to <角色3>[,<用户1>]
[with admin option]
```

4. 角色权限收回

```SQL
revoke <权限>[,<权限>]...
on <对象类型> <对象名>
from <角色>[,<角色>]...
```

4.2.6 强制存取控制方法

- 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。
- 主体：系统中的活动实体，包括实际用户和用户的各进程
- 客体：系统中的被动实体，包括文件，表，索引等
- 敏感度标记：绝密→机密→可信→公开
- 主体的敏感度标记：许可证级别
- 客体的敏感度标记：密级

4.3 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

4.4 审计

- 审计是数据库管理系统达到c2以上安全级别必不可少的一项指标。
- 审计功能把用户对数据库的所有操作自动记录下来放入审计日志，审计员可以利用审计日志监控数据库的各种行为，重现导致现状的一系列事件。
- 审计事件：（1）服务器事件（2）系统权限（3）语句事件（4）模式对象事件

- 审计功能：（1）基本功能（2）多套审计规则（3）审计分析和报表（4）审计日志管理（5）查询审计设置及审计记录信息的专门视图
- AUDIT语句和NOAUDIT语句

```SQL
audit alter,update
on SC;

noaudit alter,update
on SC;
```

4.5 *数据加密

- 加密的基本思想是根据一定算法将**明文**变换为**密文**
- 数据加密包括：
  （1）存储加密
  （2）传输加密

4.6 *其他安全性保护

- 除**自主存取控制**和**强制存取控制**外，还有**推理控制**，以及数据库应用中**隐蔽信道**和**数据隐私保护**等技术

第五章 数据库完整性

- 数据库的**完整性**是指数据的**正确性**和**相容性。**
  （1）正确性：数据符合现实世界语义
  （2）相容性：同一对象在不同关系表中符合逻辑
- 注意完整性和安全性的联系与区别！
- 如何维护完整性：
  （1）提供定义完整性约束条件的机制
  （2）提供完整性检查的方法
  （3）进行违约处理

5.1 实体完整性

5.1.1 定义实体完整性

- 主码不能为空

```SQL
create table student
(no char(9) primary key,     /*列级定义主码*/
name char(20) not null
);
create table student
(no char(9),
name char(20) not null,
primary key (no)             /*表级定义主码*/
);
```

- **定义属性组为主码只能在表级定义**

5.1.2 实体完整性检查和违约处理

1. 检查主码值是否唯一
2. 检查主码的各个属性是否为空

5.2 参照完整性

5.2.1 定义参照完整性

- 使用references 指明外码参照的主码

```SQL
create table sc
 (no char (9)  not null,
name char (20) not null,
grade smallint,
primary key (no, name)
foreign key (no) references student (no) ,
) ;
```

5.2.2 参照完整性检查和违约处理

1. 拒绝操作
2. 级联操作
3. 设置为空值

5.3 用户定义的完整性

5.3.1 属性上的约束条件的定义

- 包括：
  （1）列值非空（not null）
  （2）列值唯一（unique）
  （3）检查是否满足一个表达式（check）

```SQL
create table sc
(no char(9) unique not null,
name char(20),
sex char(2) check(sex in ('男','女'))
);
```

5.3.2 元组上的约束条件

- 定义：

```SQL
create table sc
(no char(9) not null,
name char(20),
sex char(2),
primary key (no),
check (sex='女' or name not like 'Ms.%')
);
```

5.4 完整性约束命名子句

1. 完整性约束名命子句

```SQL
create table student
 (
no char(9)
  constraint C1 check (no between 90000 and 99999),
name char(20)
  constraint C2 check not null
) ;
```

- constraint <完整性约束条件名> <完整性约束条件>

2. 修改表中的完整性约束条件：

```SQL
alter table student
    drop constraint C1;
alter table student
    add constraint C2 check(no between 100 and 100000)
```

5.6 断言

- 通过声明性断言来指定更具一般性的约束。

1. 创建断言的语句格式

```SQL
create assertion <断言名> <check 子句>
```

5.7 *触发器

- 触发器：是用户定义在关系表上的一类由事件驱动的特殊过程。

5.7.1 定义触发器

- 触发器又叫事件-条件-动作 规则
- 一般格式：

```SQL
CREATE TRIGGER<触发器名>   /*每当触发事件发生时,该触发器被激活*/
{BEFORE| AFTER}<触发事件>ON<表名>
                          /*指明触发器激活的时间是在执行触发事件前或后*/
REFERENCING NEWOLD ROW AS<变量>
                              /* REFERENCING指出引用的变量*/
FOR EACH {ROW I STATEMENT}
                              /*定义触发器的类型,指明动作体执行的频率*/
[ WHEN <触发条件>]<触发动作体>
                              /*仅当触发条件为真时才执行触发动作体*/
```

> (1)只有表的拥有者,即创建表的用户才可以在表上创建触发器,并且一个表上只能创建一定数量的触发器。触发器的具体数量由具体的关系数据库管理系统在设计时确定。

> (2)触发器名
> 触发器名可以包含模式名,也可以不包含模式名。同一模式下,触发器名必须是唯一的,并且触发器名和表名必须在同一模式下。
> (3)表名
> 触发器只能定义在基本表上,不能定义在视图上。当基本表的数据发生变化时,将激活定义在该表上相应触发事件的触发器,因此该表也称为触发器的目标表
> (4)触发事件
> 触发事件可以是 INSERT、 DELETE或 UPDATE,也可以是这几个事件的组如
> INSERT OR DELETE等,还可以是 UPDATE OF<触发列,...>,即进一步指明修改哪些列时激活触发器。 AFTER/BEFORE是触发的时机。 AFTER表示在触发事件的操作执行之后激活触发器; BEFORE表示在触发事件的操作执行之前激活触发器。
> (5)触发器类型
> 触发器按照所触发动作的间隔尺寸可以分为行级触发器(FOR EACH ROW)和语句级触发器(FOR EACH STATEMENT)
> 6)触发条件
> 触发器被激活时,只有当触发条件为真时触发动作体才执行,否则触发动作体不执行。如果省略WHEN触发条件,则触发动作体在触发器激活后立即执行。
> (7)触发动作体
> 触发动作体既可以是一个匿名 PLSQL过程块也可以是对已创建存储过程的调用,如果是行级触发器,用户可以在过程体中使用NEW和OLD引用 UPDATE/INSERT事件之后的新值和 UPDATE/DELETE事件之前的旧值;如果是语句级触发器,则不能在触发动作体中使用NEW或OLD进行引用。
> 如果触发动作体执行失败,激活触发器的事件(即对数据库的增、删、改操作)就会终止执行,触发器的目标表或触发器可能影响的其他对象不发生任何变化。

```SQL
create trigger insert_or_update_sal /*插入或更新时激活触发器*/
before insert or update on Teacher /*before触发事件*/
referencing new row as newTuple
for each row                              /*这是行级触发器*/
begin                                       /*定义触发动作体*/
    if(newtuple.Job='教授')and(newtuple.Sal<4000)
        then newtuple.Sal :=4000;
    end if;
end;                                          /*触发动作体结束*/
```

- 删除触发器
  drop trigger <触发器名> on <表名>;

第六章 关系数据理论

6.1 问题的提出

- 1NF：每一个分量必须时不可分的数据项
- 关系是关系模式在某一时刻的状态或内容
- 关系模式是静态的

6.2 规范化

6.2.1 函数依赖

1. **函数依赖**:
   某个属性集决定另一个属性集时，称另一属性集依赖于该属性集，Y函数依赖于X，X函数决定Y记作：X→Y
2. 函数依赖属于语义范畴
3. 非平凡的函数依赖：X→Y，但Y不包含于X
4. 平凡的函数依赖：X→Y，但Y包含于X
5. 若X→Y，则称X为**决定因素**
6. **完全函数依赖**：若X→Y，并且对于任意X的一个真子集x'，都有Y不函数依赖于x',则称Y完全函数依赖于X，
   否则为**部分函数依赖**
7. 传递函数依赖:
   若X→Y（Y不包含于X），Y-/->X，Y→Z（Z不包含于Y），则称 Z对X**传递函数依赖**

- 当B属性函数依赖于A属性是，属性A对B的联系是 **多对1** .

6.2.2 码

1. 设K是R<U，F>中的属性或属性组，若U完全函数依赖于K，则K为R的候选码。
2. 如果U函数依赖于K，即：K→U，则K称为超码
3. 包含在任何一个候选码中的属性称为主属性
   ，否则为非主（码）属性
4. 整个属性组是码，称为全码

6.2.3 范式

- 规范化：一个低一级范式的关系模式通过模式分解可以转化为若干个高一级范式的关系模式的集合
- 规范化的过程主要是为了克服：**插入异常，删除异常，修改复杂**

1. **1NF**：每一个分量必须是不可分的数据项
2. **2NF**：若R∈1NF，且每一个非主属性完全函数依赖于任何我一个候选码
3. **3NF**：在2NF基础上，任何非主属性不依赖于其它非主属性(在2NF基础上消除传递依赖)
4. **BCNF**:
   （1）每一个决定因素都包含码
   （2）所有非主属性对每一个码都是完全函数依赖
   （3）所有主属性对每一个不包含它的码也是完全函数依赖
   （4）没有任何属性完全函数依赖于非码的任何一组属性
5. 多值依赖:
   设R(U)是一个属性集U上的一个关系模式， X、 Y和Z是U的子集，并且Z = U - X - Y，多值依赖 X→→Y成立当且仅当对R的任一关系r，r在（X，Z）上的每个值对应一组Y的值，这组值仅仅决定于X值而与Z值无关

- 多值依赖具有对称性，传递性

6. **4NF**：关系模式R<U，F>∈1NF，如果对于R的每个非平凡多值依赖X→→Y（Y不包含于X），X都含有候选码，则R∈4NF。

- 4NF就是限制关系模式的属性之间不允许有非平凡且非函数依赖的多值依赖

6.3 数据依赖的公理系统

1. **逻辑蕴含**
对于满足一组函数依赖 F 的关系模式R <U，F>，其任何一个关系r，若函数依赖X→Y都成立, 则称 F逻辑蕴含X →Y .
2. **Armstrong 公理系统**:
关系模式R <U，F >来说有以下的推理规则:
Al.自反律（Reflexivity）： 若Y ⊆ X ⊆ U，则X →Y为F所蕴含。
A2.增广律（Augmentation）：若X→Y为F所蕴含，且Z ⊆ U，则XZ→YZ为F所蕴含。
A3.传递律（Transitivity）：若X→Y及Y→Z为F所蕴含，则X→Z为F所蕴含。

- 注意：由自反律所得到的函数依赖均是平凡的函数依赖，自反律的使用并不依赖于F
- 根据A1，A2，A3这三条推理规则可以得到下面三条**推理规则**:
合并规则：由X→Y，X→Z，有X→YZ。
（A2， A3）
伪传递规则：由X→Y，WY→Z，有XW→Z。
（A2， A3）
分解规则：由X→Y及 Z ⊆ Y，有X→Z。
（A1， A3）

3. **闭包**:
(1) 在关系模式R<U，F>中为F所逻辑蕴含的函数依赖的全体叫作 F的闭包，记为F+。
(2) 设F为属性集U上的一组函数依赖，X ⊆ U， XF+ ={ A|X→A能由F 根据Armstrong公理导出}，XF+称为属性集X关于函数依赖集F 的闭包。

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

4. **最小依赖集**:

- 覆盖：如果G+=F+，就说函数依赖集F覆盖G（F是G的覆盖，或G是F的覆盖），或F与G等价。
- F+ = G+ 的充分必要条件是： F ⊆ G+ ，和G ⊆ F+
- 如果函数依赖集F满足下列条件，则称F为一个极小函数依赖集。亦称为最小依赖集或最小覆盖。
(1) F中任一函数依赖的右部仅含有一个属性。
(2) F中不存在这样的函数依赖X→A，使得F与F-{X→A}等价。
(3) F中不存在这样的函数依赖X→A， X有真子集Z使得F-{X→A∪{Z→A}与F等价。

6.4 模式分解

- 模式分解的等价定义
（1.）分解具有无损连接性
（2.）分解要保持函数依赖
（3.）分解既要保持函数依赖，又要具有无损连接性

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

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

- 设关系模式R<U,F>被分解为若干个关系模式
R1<U1,F1>，R2<U2,F2>，…，Rn<Un,Fn>
（其中U=U1∪U2∪…∪Un，且不存在Ui ⊆ Uj，Fi为F在Ui上的投影）。
若F所逻辑蕴含的函数依赖一定也由分解得到的某个关系模式中的函数依赖Fi所逻辑蕴含，则称关系模式R的这个分解是保持函数依赖的(Preserve Dependency)

- 判别一个分解的无损连接性

- 转换为3NF的保持函数依赖的分解。（合成法）

- 转换为3NF既有无损连接性又保持函数依赖的分解

- 转换为BCNF的无损连接分解（分解法）

- 达到4NF的具有无损连接性的分解

- 对关系模式进行分解时，要求保持**函数依赖**，最高可以达到 **3NF**。

- 关系模式分解为BCNF后，**函数依赖关系可能被破坏。**

- 模式分解具有无损连接性和保持函数依赖的**两个互相独立的标准。**具有无损连接性的分解不一定保持函数依赖，保持函数依赖的分解不一定具有无损连接性

第七章 数据库设计

7.1 数据库设计概述

1. 数据库设计：数据库设计是指对于一个给定的应用环境，构造（设计）优化的数据库逻辑模式和物理结构，并据此建立数据库及其应用系统，使之能够有效的存储和管理数据，满足各种用户的应用需求，包括信息管理要求和数据操作要求。

7.1.1 数据库设计的特点

1. 数据库建设的基本规律：
   “三分技术，七分管理，十二分基础数据”

- 强调了数据的收集，整理，组织和不断更新是数据库建设中的重要一环。
  2.结构（数据）设计和行为（处理）设计相结合

7.1.2 数据库设计方法

- 计算机的基础知识
- 软件工程的原理和方法
- 程序设计的方法和技巧
- 数据库的基础知识
- 数据库设计技术
- 应用领域的知识

7.1.3 数据库设计的基本步骤

1. 需求分析：最困难，最耗时，设计过程的基础
2. 概念设计：数据库设计的关键
3. 逻辑结构设计：
4. 物理结构设计：
5. 数据库实施
6. 数据库运行和维护

设计阶段	设计描述
需求分析	数据字典，全系统数据项，数据结构，数据流，数据存储的描述
概念结构设计	概念模型（E-R图），数据字典
逻辑结构设计	某种数据模型
物理结构设计	存储安排，存取方法选择，存取路径建立
数据库实施	创建数据库模式，数据载入，应用程序的编码和调试，运行
数据库运行和维护	性能监测，转储/恢复，数据库重组，重构

7.2 需求分析

7.2.1 需求分析的任务

- 详细调查现实世界要处理的对象，充分了解工作概况，明确用户各种需求，在此基础上确定功能。
- 调查重点是数据和处理，通过调查，收集，分析，获取用户对数据库的如下要求：
  （1）信息要求（内容和性质）
  （2）处理要求（处理性能）
  （3）安全性与完整性要求

7.2.2 需求分析的方法

1. 调查机构情况
2. 调查业务活动
3. 协助用户明确要求
4. 确定新系统边界

7.2.3 数据字典

- 数据字典是进行详细的数据收集和数据分析所获得的主要成果。
- 数据字典是关于数据库中数据的描述，即元数据，而不是数据本身。
- 数据字典是在需求分析阶段建立，在数据库设计过程中不断修改，充实，完善的，在数据库设计中占有很重要的地位
- 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。
  （1）数据项是**数据的最小组成单位**，数据项是**不可再分的数据单位**，若干数据项可以组成数据结构。
  （2）数据结构反映了数据之间的组合关系
  （3）数据流是数据结构在系统内传输的路径
  （4）数据存储是数据结构停留或保存的地方，也是数据流的来源和去向之一。
  （5）处理过程的具体处理逻辑一般用判定表，判定树来描述

7.3 概念结构设计

7.3.1 概念模型

- 特点：
  （1）真实，充分反映现实世界
  （2）易于理解
  （3）易于更改
  （4）易于向关系，网状，层次等转化

7.3.2 E-R模型

- 从数据流图构造E-R图时，选择实体一般应先考虑数据流图中的**数据存储**。

1. 实体之间的联系：1对1，1对多，多对多
2. E-R图：
   （1）实体型用矩形

（2）属性用椭圆形
（3）联系用菱形

3. 作为属性，不能再具有需要描述的性质
4. 属性不能与其他实体具有联系
5. E-R图之间的冲突有三类：
（1）属性冲突：①属性域冲突②属性取值单位冲突
（2）命名冲突：①同名异义②异名同义
（3）结构冲突：
- ①同一对象在不同应用种具有不同的抽象
- ②同一实体在不同子系统的E-R图中所包含的属性个数和属性排列次序不完全相同
- ③实体间的联系在不同的E-R图中为不同的类型。

7.4 逻辑结构设计

7.4.1 E-R图向关系模型的转化

- 1:1联系的转换方法
联系转换为一个独立的关系：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，且每个实体的码均是该关系的候选码。
将1:1联系与某一端实体集所对应的关系合并，则需要在被合并关系中增加属性，其新增的属性为联系本身的属性和与联系相关的另一个实体集的码。

> 联系形成的关系独立存在：
> 职工表（职工号，姓名，年龄）主码：职工号
> 产品表（产品号，产品名，价格）主码：产品号
> 负责（职工号，产品号）主码：职工号或产品号

- 1:n联系的转换方法
一种方法是将联系转换为一个独立的关系，其关系的属性由与该联系相连的各实体集的码以及联系本身的属性组成，而该关系的码为n端实体集的码；另一种方法是在n端实体集中增加新属性，新属性由联系对应的1端实体集的码和联系自身的属性构成，新增属性后原关系的码不变

> 联系形成的关系独立存在。
> 仓库（仓库号，地点，面积）主码：仓库号
> 产品（产品号，产品名，价格）主码：产品号
> 仓储（仓库号，产品号，数量）主码：产品号
> 合并后方案：联系形成的关系与n端对象合并。
> 仓库（仓库号，地点，面积）
> 产品（产品号，产品名，价格，仓库号，数量）

- m:n联系的转换方法
在向关系模型转换时，一个m:n联系转换为一个关系。转换方法为：与该联系相连的各实体集的码以及联系本身的属性均转换为关系的属性，新关系的码为两个相连实体码的组合（该码为多属性构成的组合码）。

> 选课
> •该模型包含两个实体集（学生、课程）和一个m:n联系
> •该模型可转换为三个关系模式：
> 学生（学号，姓名，性别，年龄）主码：学号
> 课程（课程号，课程名，学分）主码：课程号
> 选课（学号，课程号，成绩）主码：学号+课程号

7.4.2 数据模型的优化

- 并不是规范化程度越高的关系就越优

7.4.3 设计用户子模式

7.5 物理结构设计

- 物理结构设计：为一个给定的逻辑数据模型选取一个最适合应用要求的物理结构的过程
- 步骤：（1）确定数据库的物理结构（2）对物理结构进行评价

7.5.1 数据库物理设计的内容和方法

- 内容主要包括：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

7.5.2 关系模式存取方法选择

- 常用方法：索引方法和聚簇方法
（1）B+树索引存取方法：索引并不是越多越好，会降低更新的效率
（2）hash索引存取方法：选择条件①关系的大小不可知②关系大小动态改变
（3）聚簇存取方法：
- 聚簇：把这个或这些属性上具有相同值的元组集中放在连续的物理块中
- 该属性（组）称为聚簇码
- 建立和维护聚簇的开销相当大。聚簇码值要相对稳定，以减小开销

7.5.3确定数据库的存储结构

- 考虑：存取时间，存储空间利用率，维护代价

1. 确定数据存放位置
2. 确定系统配置

7.6 数据库的实施和维护

7.6.1 数据的载入和应用程序的调试

- 两项重要工作：数据的载入，应用程序的编码和调试
- 数据库应用程序的设计应该与数据库设计同时进行

7.6.2 数据库的试运行

- 先输入小批量数据作调试用，带试运行基本合格后再大批量输入数据
- 做好数据库的转储和恢复工作

7.6.3 数据库的运行和维护

1. 数据库的转储和恢复
2. 数据库的安全性，完整性控制
3. 数据库性能的监督，分析和改造
4. 数据库的重组织与重构造

- 重组织不修改原设计的逻辑和物理结构
- 重构造部分修改数据库的模式和内模式

第八章 数据库编程（无）

第九章 关系查询处理和查询优化

9.1 关系数据库系统的查询处理

9.1.1 查询处理步骤

- 四个阶段：**查询分析，查询检查，查询优化，查询执行**

1. 查询分析：**语法检查**，对查询语句进行扫描，词法分析，语法分析
2. 查询检查：
   （1）**语义检查**，根据数据字典中的模式定义和用户权限和完整性约束定义
   （2）检查通过后转化为等价的关系代数表达式
   （3）关系数据库管理系统一般都用**查询树**，也叫**语法分析树**
3. 查询优化：
   （1）按照优化层次分为：代数优化，物理优化
   （2）代数优化策略是通过对关系代数表达式的**等价变换**来提高查询效率。
   （3）物理优化则是指存取路径和底层操作算法的选择。选择的依据：①基于规则②基于代价③基于语义
4. 查询执行：依据优化器得到的执行策略生成查询执行计划

9.1.2 实现查询操作

1. 选择操作的实现：一般采用全表扫描或者基于索引的算法
2. 连接操作的实现：是查询处理中最常用也最耗时的操作之一，一般采用：嵌套循环算法，排序-合并算法，索引连接算法，hash join算法

9.2 关系数据库系统的查询优化

- 查询优化的优点不仅在于用户不必考虑如何更好的表达查询以获得较高效的效率，而且在于系统可以比用户程序的优化做得更好

9.3 代数优化

9.3.1 关系代数表达式等价变换规则

- 代数优化策略是通过对关系代数表达式的**等价变换**来提高查询效率。
- 常用代数优化策略：
  （1）连接，笛卡儿积的交换律
  （2）连接，笛卡儿积的结合律
  （3）投影的串接定律
  （4）选择的串接定律
  （5）选择与投影操作的交换律
  （6）选择与笛卡儿积的交换律
  （7）选择与并的分配律
  （8）选择与差运算的分配律
  （9）选择对自然连接的分配律
  （10）投影与笛卡儿积的分配律
  （11）投影与并的分配律

9.3.2 查询树的启发式优化

1. 选择运算应尽可能先做
2. 把投影运算和选择运算同时进行
3. 把投影同其前或后的双目运算结合
4. 把某些选择同它前面要执行的笛卡儿积结合起来成为一个连接运算
5. 找出公共子表达式

9.4 物理优化

- 选择的方法：（1）基于规则的启发式优化（2）基于代价估算的优化（3）两者结合的优化

第十章 数据库恢复技术

10.1 事务的基本概念

1. 事务

- 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**
- 事务可以是一条或多条sql语句，一个或多个程序
- 定义事务的语句：
  （1）begin transaction
  （2）commit
  （3）rollback
- 事务以begin transaction 开始，以commit或rollback结束
- commit 表示提交事务的所有操作，将事务中所有对象对数据库的更新写回到磁盘上的物理数据库中去
- rollback 表示回滚，系统将事务中对数据库的所有已完成的操作全部撤销，回滚到事务开始时的状态

2. 事务的ACID特性

- 四个特性：原子性，一致性，隔离性，持续性
- 原子性：事务是数据库的逻辑工作单位
- 一致性
- 隔离性：一个事务的执行不被其他事务打扰
- 持续性：**事务一旦提交，改变即永久**，也叫永久性

10.2 数据库恢复概述

- 从错误状态恢复到某一已知正确状态

10.3 故障的种类

1. **事务故障**：例如：并发事务发生**死锁**，**违反完整性**约束而被终止，**运算溢出**
   （1）多是非预期的，不能由应用程序处理的。
   （2）这类恢复操作成为事务撤销
2. **系统故障**：例如：**cpu故障，os故障，DBMS代码故障，断电**
   （1）造成系统停止运转的任何事件
   （2）需要**重做**所有已提交的事务
3. **介质故障**：例如：**磁盘损坏，磁头碰撞，瞬时强磁场干扰**

- 系统故障称为软故障，介质故障称为硬故障，如外存故障

4. **计算机病毒**

- 恢复的基本原理：**数据冗余**

10.4 恢复的实现技术、

- 建立**冗余数据**最常用的技术就是**数据转储**和**登记日志文件**

10.4.1 数据转储

- 转储：数据库管理员定期地将数据库复制到其他存储介质上保存起来
- 这些备用的数据叫做：**后备副本**
- 转储分为静态转储和动态转储，还可分为海量转储和增量转储
- 为恢复到某一时刻的正确状态需要把转储期间的各事务对数据库的修改活动登记下来，即建立：**日志文件**

10.4.2 登记日志文件

1. 日志文件：用来记录事务对数据库的更新操作的文件
2. 登记内容：事务的开始，结束，所有更新操作
3. 内容：事务标记，操作类型，操作对象，更新前旧值，更新后新值
4. 作用：事务故障恢复，系统故障恢复，协助后备副本进行介质故障恢复
5. 登记日志文件注意：
   （1）登记次序严格按并发事务执行的事件次序
   （2）必须先写日志文件，后写数据库

10.5 恢复策略

10.5.1 事务故障的恢复

1. 务故故障是指事务在运行至正常终止点前被终止,这时恢复子系统应**利用日志文件撤销(UNDO)此事务已对数据库进行的修改。**事务故障的恢复是由系统自动完成的,对用户是透明的。

- 系统的恢复步骤是:
  (1)反向扫描日志文件(即从最后向前扫描日志文件),查找该事务的更新操作。
  (2)对该事务的更新操作执行逆操作,即将日志记录中“更新前的值”写入数据库
  (3)继续反向扫描目志文佚,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。

10.5.2 系统故障的恢复

- 系统故障造成数据库不一致状态的原因有两个,一是未完成事务对数据
  库的更新可能已写入数据库,二是已提交事务对数据库的更新可能还留在缓冲区没来得及写入数据库。因此恢复操作就是要**撤销故障发生时未完成的事务,重做已完成的事务。**
- 系统故障的恢复是由系统在重新启动时自动完成的,不需要用户干预。

- 系统的恢复步骤是:
(1)正向扫描日志文件(即从头扫描日志文件),找出在故障发生前已经提交的事务，将其事务标记入重做队列(REDO-LIST)，同时找出故障发生时尚完成的事务（只有BEGIN 无commit），将其事务标识记撤销队列(UNDO-LIST)。
(2)对撤销队列中的各个事务进行撤销(UNDO)处理
进行撤销处理的方法是,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)对重做队列中的各个事务进行重做处理
进行重做处理的方法是:正向扫描日志文件,对每个重做事务重新执行日志文件登记的操作,即将日志记录中“更新后的值”写入数据库。

10.5.3 介质故障的恢复

- 发生介质故障后,磁盘上的物理数据和日志文件被破坏,这是最严重的一种故障,恢复方法是**重装数据库,然后重做已完成的事务。**
(1)装入最新的数据库后备副本(离故障发生时刻最近的转储副本),使数据库恢复到最近一次转储时的一致性状态
- 对于动态转储的数据库副本,还需同时装入转储开始时刻的日志文件副本,利用恢复系统故障的方法(即REDO+UNDO),才能将数据库恢复到一致性状态。
(2)装入相应的日志文件副本(转储结束时刻的日志文件副本),重做已完成的
即首先扫描日志文件,找出故障发生时已提交的事务的标识,将其记入重做队列:然后正向扫描日志文件,对重做队列中的所有事务进行重做处理。即将日志记录中“更新后的值”写入数据库。这样就可以将数据库恢复至故障前某一时刻的一致状态了。
- 介质故障的恢复需要数据库管理员介入,但数据库管理员只需要重装最近转储的数据库副本和有关的各日志文件副本,然后执行系统提供的恢复命令即可,具体的恢复操作仍由数据库管理系统完成。

10.6 具有检查点的恢复技术

- 记录检查点(checkpoint)记录：增加一个重新开始文件,并让恢复子系统在登录日志文件期间动态地维护日志
- 检查点记录内容：
（1）建立检查点时刻所有正在执行的事务清单
（2）这些事务最近一个日志记录的地址
- 动态维护日志文件方法：周期性地执行建立检查点保存数据库状态的操作
- 使用检查点的方法可以改善恢复效率

10.7 *数据库镜像

- 通过复制数据实现，用于数据库恢复

第十一章 并发控制

- 事务可以一个一个地串行执行，即每个时刻只有一个事务运行
- 多处理机下，事务可以实现多个事务并行运行，即同时并发方式
- 为保证多用户环境中数据的完整性和一致性，DBMS采用的控制称为**并发控制**

11.1 并发控制概述

1. **事务**是并发控制的基本单位
2. 并发控制是对用户的**并发操作**加以控制和协调
3. 并发操作造成 不一致性 的问题包括：
（1）**丢失修改**：
- 考虑飞机订票系统中的一个活动序列:

> 甲售票点（甲事务）读出某航班的机票余额A,设A=16.
> 乙售票点（乙事务）读出同一航班的机票余额A,也为16.
> 甲售票点卖出一张机票,修改余额A←A-1.所以A为15,把A写回数据库.
> 乙售票点也卖出一张机票,修改余额A←A-1.所以A为15,把A写回数据库.
> 结果明明卖出两张机票，数据库中机票余额只减少1。
> 归纳起来就是：两个事务T1和T2读入同一数据并修改，T2提交的结果破坏了T1提交的结果，导致T1的修改被丢失。

（2）**不可重复读**：

- 不可重复读是指事务T1读取数据后，事务T2执行更新操作，使T1无法再现前一次读取结果。具体地讲，不可重复读包括三种情况：
事务T1读取某一数据后，事务T2对其做了修改，当事务1再次读该数据时，得到与前一次不同的值。

> 例如，T1读取B=100进行运算，T2读取同一数据B，对其进行修改后将B=200写回数据库。T1为了对读取值校对重读B，B已为200，与第一次读取值不一致。
> 事务T1按一定条件从数据库中读取了某些数据记录后，事务T2删除了其中部分记录，当T1再次按相同条件读取数据时，发现某些记录神密地消失了。
> 事务T1按一定条件从数据库中读取某些数据记录后，事务T2插入了一些记录，当T1再次按相同条件读取数据时，发现多了一些记录。（这也叫做幻影读）

（3）**读“脏”数据**：

- 读"脏"数据是指事务T1修改某一数据，并将其写回磁盘，事务T2读取同一数据后，T1由于某种原因被撤消，这时T1已修改过的数据恢复原值，T2读到的数据就与数据库中的数据不一致，则T2读到的数据就为"脏"数据，即不正确的数据。

4. 产生上述数据不一致性的**主要原因**是，**并发操作破坏了事务的隔离性**

- 产生数据不一致性的**根本原因**是：**数据冗余**

5. 并发控制的主要技术：
（1）封锁
（2）时间戳
（3）乐观控制法
（4）多版本并发控制

11.2 封锁

1. 封锁是实现并发控制的一个重要技术
2. 封锁的类型：**排他锁（写锁），共享锁（读锁）**

11.3 封锁协议

- 三级封锁协议

11.4 活锁和死锁

11.4.1 活锁

- 避免活锁的简单方法是采用**先来先服务**的策略

11.4.2 死锁

1. 死锁的预防：
   （1）一次封锁法
   （2）顺序封锁法
2. 死锁的诊断与解除：
   诊断：
   （1）超时法
   （2）等待图法（DBMS普遍采用）
   解除：选择一个代价最小的事务，进行撤销

11.5 并发调度的可串行性

- 可串行化调度：多个事务的并发执行是正确的，当且仅当其结果与按某一次序串行地执行这些事务时的结果相同。
- 可串行性：是并发事务正确调度的准则
- **冲突可串行化调度**是可串行化调度的**充分条件**
- **事务遵守两段锁协议**是可串行化调度的**充分条件**
- 冲突操作是指不同的事物对同一数据的读写操作和写写操作
- 遵守两段锁协议的事务也可能会死锁

11.6 两段锁协议

- 第一阶段：封锁（扩展阶段）
- 第二阶段：释放封锁（收缩阶段）
- 事物遵守**两段锁协议**是可串行化调度的充分条件，而不是必要条件

11.7 封锁的粒度

- 封锁粒度：封锁对象的大小
- 封锁对象可以是逻辑单元，也可以是物理单元
- 封锁粒度与系统的**并发度**和并发控制的**开销**密切相关
- 多粒度封锁：
  （1）显式封锁
  （2）隐式封锁
- 意向锁：如果对一个结点加意向锁，则说明该节点的下层结点正在被加锁。
  分类：
  （1）意向共享锁（IS锁）
  （2）意向排他锁（IX锁）
  （3）共享意向排他锁（SIX锁）

第十二章 *数据库管理系统（无）