

# 2018 年版 数据库系统原理

04735

学校：邮 电

作者：马明洋

计算机信息管理：20230929

# 数据库系统原理 04735

## 1. 数据库系统概述

### 1.1. 基本概念

1.1.1. 数据：描述事物的符号记录

1.1.2. 数据库：长期存储 有组织, 可共享 的数据集合

#### 1.1.3. 数据库管理系统

专门用于建立和管理数据库的一套软件

快速获取所需数据, 提供安全性和完整性统一控制机制, 对数据有效管理和维护

1) 数据定义功能：定义语言, 表视图等 ; DDL

2) 数据操纵功能：插入 删除 查询 修改 ; DML

3) 数据库的运行管理功能：多用户共享, 安全性, 不干扰并发, 统一管理, 恢复

4) 数据库的建立和维护功能：创建数据库, 数据库空间维护, 备份恢复

5) 数据组织, 存储和管理功能：提高数据存储效率, 对数据进行分类存储和管理

6) 其他功能：通过接口相互访问

#### 1.1.4. 数据库系统 ( 数据库系统 >>> 数据库 )

包括:

1) 数据库

2) 数据库管理系统

3) 工具, 应用程序, 数据库管理员, 用户

### 1.2. 数据管理技术发展

业务管理：负责制定并执行组织中对数据的定义, 组织, 保护; 有效使用策略, 过程, 计划

依靠技术：负责实现数据作为一种资源的集中管理控制

#### 1) 人工管理阶段：

20 世纪 50 年代中期前 ( 1950~5 前 ) 手工无磁盘, 数据和应用一对一

#### 2) 应用程序管理数据：

20 世纪 50 年代后期到 60 年代中期 ( 1955 -> 1965~9 )

文件系统存储在磁盘通过抽取排序合并对应用提供新文件

#### 3) 数据库系统阶段：

20 世纪 60 年代后期以来 ( 1965~9 -> now )

(1) 数据集成: 逻辑联系, 数据被组织统一逻辑结构, 与物理组织定位分离

(2) 数据共享性高: 一个数据可被多个用户共同使用

(3) 数据冗余小: 一个数据项(字段)可以理想的只保存一次, 节省空间

(4) 数据一致性: 可以通过更新副本保持数据一致

(5) 数据独立性高: 数据定义与使用的应用程序分离, 不同用户不同视图

(6) 实施统一管理和控制:

① 安全性: 保护数据, 防止不合法操作对数据的损坏, (权限)

② 完整性: 包含 正确性, 有效性, 相容性 ; 满足约束条件

③ 并发控制: 避免多用户同时存取修改数据库时发生干扰导致错误结果

④ 故障恢复: 将快照备份将数据库恢复到正确状态

(7) 减少应用程序开发和维护工作量: 共享性, 独立性

### 1.3. 数据库系统的结构

从数据库管理员看:可分为内部系统结构和外部系统结构, **内部**通常采用**三级模式**结构  
**外部体系**结构通常表现为 **集中式, 分布式 并行**结构

#### 1.3.1. 三级模式

用户级, 概念级, 物理级; 数据库系统是由**模式, 外模式, 内模式** 三级构成

**1. 模式** : 也称**概念模式**或**逻辑模式**;全数据逻辑结构和特征描述;全用户公共视图

- 1) 定义数据的逻辑结构, 数据之间的联系 **DDL 模式 概念或逻辑模式**
- 2) 数据相关的安全性, 完整性 等要求
- 3) **按外模式项用户提供数据, 按内模式存储数据**

**2. 外模式** : 用户模式或子模式, 是用户可以使用的局部数据逻辑结构和特征描述

- 1) 满足不同用户需求的数据视图, 即用户视图; 通常是模式的子集**局部重构**
- 2) 保证数据库安全的重要措施 **DDL 子模式 用户模式**

**3. 内模式** : 也称存储模式, 数据库内部结构的表现形式, 内部视图或存储视图

- 1) 一个数据库只有一个内模式 **DDL 内模式 存储模式**
- 2) 定义所有内部记录类型, 索引和文件的组织方式; 存储结构记录

#### ● **外模式/模式映像**

模式可以有多个外模式, 一个外模式只能对应一个外模式/模式映像, 实现数据独立性

#### ● **模式/内模式映像**

定义了数据库全局逻辑结构和物理存储, 数据库只有一个内模式, 也只有一个模式/内模式映像, 实现数据存储独立性

#### 1.3.2. 运行与应用结构

1. 客户/服务器结构
2. 浏览器/服务器结构

### 1.4. 数据模型

#### 1.4.1. 数据特征与数据模型组成要素

1. **数据结构**: 层次结构/模型, 网状结构/模型, 关系结构/模型
2. **数据操作**: 更新 和 检索, 包括插入删除修改
3. **数据约束**: 保证数据的**正确性, 有效性, 相容性**

#### 1.4.2. 数据模型的分类

##### 1. 概念层数据模型 (设计阶段)

(1) **概念模型表示方法 E-R 图** 第三章详解 E-R 图

实体型--矩形      属性--椭圆

联系--菱形 (1:1   1:N   M:N)

##### 2. 逻辑层数据模型

###### (1) **层次模型**

只有一个根结点 其他节点只有一个父结点 难以表达实体关系

###### (2) **网状模型**

允许有多父结点, 也允许多个结点没有父结点 方便的表示关系, 但结构复杂

###### (3) **关系模型**

表, 具有更高的数据独立性, 更好的安全保密, 简化开发工作

#### (4) 面向对象模型

即是概念模型又是逻辑模型, 表达能力丰富, 对象可重复利用, 维护方便

#### 3. 物理层数据模型

数据存储模型, 对用户屏蔽

## 2. 关系数据库

### 2.1. 概述 : 关系数据模型 组织数据

### 2.2. 关系数据模型

#### 2.2.1. 关系数据结构

关系: 表

属性: 一列数据

元组: 一行数据

分量: 元组中一个属性值

码或键: 在元组唯一

超码: 在码中去除某个属性, 仍是码

候选码: 在码不能去除出某个属性, 否则不为码

主码: 若干个候选码 来确定唯一标识

主属性 和 非主属性: 包含在候选码为主, 否为非主属性

外码: 某个属性不是这个关系的主码或候选码, 而是别的关系的主码

参照关系: 外码所在关系被称为参照关系, 外码为主码称为被参照关系

域: 取值范围(性别-> 男女)

关系数据库对关系的限定:

1. 每个属性都是不可分解, 每个数据项不可分 -> 不允许表中有表
2. 每个关系只有一种关系模式, 也就是属性的数据类型及属性个数是固定的
3. 每个关系属性必须命名, 属性名不能重复
4. 同一关系不能出现候选码完全相同的元组
5. 关系中的元组和属性的顺序无要求

#### 2.2.2. 关系操作集合

1. 基本关系操作 操作的对象和结果都是集合

查询 和 插入 删除 修改 两部分

2. 关系数据语言分类

关系代数语言 关系演算语言 和具有两种特点的 SQL

3. 关系代数

关系代数表达式

#### 2.2.3. 关系完整性约束 数据库的完整性 指 正确性 相容性 一致性

- 1) 实体完整性约束: 主码不能为空
- 2) 参照完整性约束: 定义主码与外码的引用规则
- 3) 用户定义完整性约束: 用户自定义针对某一应用环境的完整性约束
- 4) 关系模型完整性约束: 使用插入删除修改操作 校验

### 2.3. 规范化理论

#### 2.3.1. 可能存在的冗余和异常

### 2.3.2. 函数依赖与关键字

完全函数依赖: (学号,课号)->成绩 学号课号决定成绩, 两个同时存在  
部分函数依赖: (学号,姓名)->性别 学号姓名都决定性别, 两个都是部分依赖  
传递函数依赖: 三个字段, 1 决定 2, 2 决定 3, 但 3 不决定 1

### 2.3.3. 范式与关系规范化过程

#### 1. 第一范式 (冗余过多)

设  $R$  为任一给定关系,若  $R$  中每个列与行的交点处的取值都是不可分割的基本元素,则  $R$  为第一范式  
不含重复组的关系(单元格合并)  
不存在嵌套结构

#### 2. 第二范式

设  $R$  为任一给定关系,若  $R$  为 1NF,且其所有非主属性都完全函数依赖于候选关键字,则  $R$  为第二范式  
课号->教师 教师->教室 教师!->课号  
有传递依赖, 其他属性聚合才能去顶候选属性(完全依赖)

#### 3. 第三范式

设  $R$  为任一给定关系,若  $R$  为 2NF,且其每一个非主属性都不传递函数依赖于候选关键字,则  $R$  为第三范式  
非主属性没有传递依赖, 每列的重复数据都不能确定其他唯一数据  
非主属性就是列有重复

#### 4. BCNF

### 2.3.4. 关系规范化理论应用

概念设计阶段,判断属性划分到那个实体中更合适

## 3. 数据库设计

### 3.1. 数据库设计概述

是将数据库系统与现实世界进行密切 有机 协调一致的结合过程

#### 3.1.1. 生命周期

数据库分析与设计阶段 数据库实现与操作阶段

#### 3.1.2. 目标

满足 应用功能需求 和 良好的数据库性能

#### 3.1.3. 内容

##### 3.1.3.1. 数据库结构设计

针对给定环境进行数据库的模式或子模式设计  
包括数据库概念结构设计,逻辑结构设计,物理结构设计  
模式定义并给出个应用程序的共享结构,静态的一但形成就不会改变

##### 3.1.3.2. 数据库行为设计

确定用户的行为和动作, 行为设计是动态的

#### 3.1.4. 方法

##### 3.1.4.1. 直观设计法: 经验和技巧

##### 3.1.4.2. 规范设计法:

##### 3.1.4.2.1. 新奥尔良设计法: 注重数据库结构设计, 不重行为设计

四阶段: 需求分析,概念结构设计,逻辑结构设计,物理结构设计



#### 3.1.4.2.2. 基于 E-R 模型的数据库设计方法: E-R 图

#### 3.1.4.2.3. 基于第三范式的设计方法

确定数据库模式,属性,属性间依赖关系

组织称一个单一关系模式中,在分析模式中不符合第三范式的约束条件  
进行模式分解,规范成若干个第三范式模式的集合

#### 3.1.4.3. 计算机辅助设计法

依据专家经验,设计过程进行辅助(case 工具)

### 3.1.5. 过程

#### 1) 需求分析阶段

#### 2) 结构设计阶段

概念结构设计,逻辑结构设计,物理结构设计

#### 3) 行为设计阶段

功能设计,事务设计,程序设计

#### 4) 数据库实施阶段

加载数据库数据,调试运行程序

#### 5) 数据库运行和维护

## 3.2. 基本步骤

### 3.2.1. 需求分析

确定数据库范围, 分析数据应用过程, 收集分析数据, 需求分析报告

### 3.2.2. 概念结构设计

在需求报告基础上按照特定的设计法满足应用需求的信息结构: 概念模型  
(E-R 图作为概念结构)

### 3.2.3. 逻辑结构设计

将概念模型转换为等价的支持数据模型的结构: 层次 网状 关系数据模型

### 3.2.4. 物理设计

对硬件读写性能了解,安排数据存放位置

### 3.2.5. 数据库实施

- 1) 加载数据: 将数据 load 到数据库中
- 2) 应用程序设计: 代码开发,处理数据程序
- 3) 数据库试运行: 测试流程,规避风险

### 3.2.6. 数据库运行和维护

长期对系统故障进行修复,或对系统进行优化效率

## 3.3. 关系数据库设计方法

关系数据库是采用关系模型作为逻辑数据模型的数据库系统  
设计过程遵从数据库设计的基本步骤 6 步

### 3.3.1. 设计过程与各级模式

概念模式, 逻辑模式, 内模式(物理设计),外模式(试运行等应用)

### 3.3.2. 概念结构设计方法

将需求分析得到的要求抽象为信息结构(概念模型)的过程  
(实体,属性,联系(一对一,一对多,多对多))

### 3.3.3. 逻辑结构设计方法

#### 1) E-R 图的关系模型转换

确定实体,属性,联系 转化为关系模式

实体,实体间的属性,实体间的联系

## **2) 数据模型的优化**

(1) 确定各属性间的函数依赖关系

(2) 对于各个关系模式之间的数据依赖进行极小化处理,消除冗余联系

(3) 判断第三范式的范式, 取最合适范式

(4) 模拟模式是否适于需求分析产生的要求,是否需分解或合并某些模式

(5) 对关系模式进行必要的分解,提高数据操作的效率和存储空间利用率

## **3) 设计用户子模式**

将概念模型转换为全局逻辑模型后

按局部要求利用视图设计符合的用户子模式

### **3.3.4. 物理设计方法**

建立索引: 对使用频率较高的属性建立

建立聚集: 将数据尽可能存放在一个块中

## **4. SQL 与关系数据库基本操作**

### **4.1. SQL 概述**

4.1.1. 发展

4.1.2. 特点

4.1.3. 组成

### **4.2. MySQL 预备知识**

4.2.1. 使用基础

4.2.2. MySQL 中的 SQL

### **4.3. 数据定义**

4.3.1. 数据库模式定义

4.3.2. 表定义

**4.3.3. 索引定义**

**Create index indexName ON DB.table(id,name);**

### **4.4. 数据更新**

4.4.1. 插入

4.4.2. 删除

4.4.3. 修改

### **4.5. 数据查询**

4.5.1. SELECT

4.5.2. 列的选择和指定

4.5.3. FROM

4.5.4. WHERE

4.5.5. GROUP BY

4.5.6. HAVING

4.5.7. ORDER BY

4.5.8. LIMIT

## 4.6. 视图

4.6.1. 创建: create or replace view viewName as select ... from ...;

4.6.2. 删除: drop view viewName;

4.6.3. 修改视图定义: alter view viewName

4.6.4. 查看视图定义: show create view viewName;

4.6.5. 更新视图数据: insert/update/delete

4.6.6. 查看视图数据: select ... from database.view;

## 5. 数据库编程

### 5.1. 存储过程

#### 5.1.1. 基本概念

- 1) 可增强 SQL 语句的功能性和灵活性
- 2) 良好的封装性
- 3) 高性能
- 4) 减少网络流量
- 5) 可作为一种安全机制确保数据库的安全性和数据的完整性

#### 5.1.2. 创建

DELIMITER \$\$ 将结束符号还为 \$\$

Create procedure plsqliName( IN idNumber INT , IN idName CHAR(10) )

BEGIN

Update table set column = idName where id=idNumber;

END \$\$

#### 5.1.3. 存储过程体

局部变量: DECLARE ID INT(10);

SET 局部变量 = "asd"; 给局部变量赋值

#### 5.1.4. 调用存储过程

调用存储过程 CALL plsqli(100,"asd");

#### 5.1.5. 删除存储过程

drop procedure plsqli;

### 5.2. 存储函数(类 java 方法)

#### 5.2.1. 创建

Create function fnName(id INT)

#### 5.2.2. 调用

Select

#### 5.2.3. 删除

Drop function fnName

## 6. 数据库安全与保护

### 6.1. 数据库完整性



### 6.1.1. 完整性约束条件的作用对象

#### (1) 列级约束

- 1) 数据类型约束,长度
- 2) 数据格式约束
- 3) 取值范围约束
- 4) 对空值的约束

#### (2) 元组约束

各个字段之间的相互约束,如开始日期要小于结束日期

#### (3) 表级约束

表中字段要符合表的含义

### 6.1.2. 定义与实现完整性约束

#### 1. 实体完整性

##### (1) 主键约束 PRIMARYKEY

- 1) 每个表只能定义一个主键
- 2) 主键值必须唯一,代表一行
- 3) 复合主键不能包含多余列,删除复合主键中一列,剩下的仍能保持唯一
- 4) 一个列名只能在复合主键中出现一次

##### (2) 候选键约束 UNIQUE

- 1) 值必须唯一,且不能为 NULL

#### 2. 参照完整性

表依赖,字段依赖

#### 3. 用户定义完整性

##### (1) 非空约束

NOT NULL

##### (2) CHECK 约束

检查限制条件

### 6.1.3. 命名完整性约束

### 6.1.4. 更新完整性约束

- (1) 完整性约束不能直接修改,智能通过 ALTER TABLE 语句先删除再重新添加
- (2) DROP TABLE 删除表,则会将表中所有约束删除

## 6.2. 触发器

是用户定义在关系表中的由事件驱动的数据库对象,任何对表的修改操作都由数据库服务器自动激活相应触发器 如: 客户每下一单,库存就要减一

### 6.2.1. 创建

```
create trigger db.triggerName after insert
On db.table for each row set @str='one table add';
```

若表有 insert 数据操作

则会在结果集显示

@str

one table add

### 6.2.2. 删除

```
drop trigger if exists db.triggerName;
```

### 6.2.3. 使用

```
create trigger db.triggerName before update
On db.table for each row
set new.column=old.column;
```

若表中有修改操作,触发器会将 column 数据更新会原来数据

## 6.3. 安全性与访问控制

### 6.3.1. 用户账号管理

#### (1) 创建用户账号

```
create user 'zhangsan'@'localhost' identified by '123456';
```

#### (2) 删除用户

```
drop user 'zhangsan'@'localhost';
```

#### (3) 修改用户

```
rename user 'zhangsan'@'localhost' to 'zhangsan'@'localhost';
```

#### (4) 修改用户口令

```
set password for 'zhangsan'@'localhost' = '123';
```

### 6.3.2. 账户权限管理

```
show grants for 'zhangsan'@'localhost';
```

#### (1) 权限授予

```
grant select(id,name) on db.table to 'zhangsan'@'localhost' = '123';
```

```
grant updata on db.table to 'zhangsan'@'localhost' ;
```

```
grant all on db.table to 'zhangsan'@'localhost' ;
```

#### (2) 权限转移

```
grant updata on db.table to 'zhangsan'@'localhost' with grant option;
```

#### (3) 权限撤销

```
revoke updata on db.table from 'zhangsan'@'localhost' ;
```

## 6.4. 事务与并发控制

### 6.4.1. 概念

sql 程序提交动作,同时运行,执行状态等

### 6.4.2. 特征

#### (1) 原子性

事务是最小的工作单位,程序要么成功要么失败,失败会撤销对数据的修改

#### (2) 一致性

一事务对数据的多条操作,要么全部成功要么全部失败,与原子性密切相关

#### (3) 隔离性

事务之间不能互相干扰,必须等待前事务成功之后执行(串行)

#### (4) 持续性

事务一旦成功,对数据的操作是永久性的

### 6.4.3. 并发操作问题

并发操作会破坏失误的隔离性,导致数据的一致性有误(封锁解决)

(1) 丢失更新

两个事务对统一数据进行修改,第二个提交会导致第一条更改数据丢失

(2) 不可重复读

次次读取数据之间,会存在其他增删修数据操作,导致数据次次不一致

(3) 读脏数据

第二事务读取时,第一事务撤销,数据恢复,导致第二事务读取数据不正确

6.4.4. 封锁

(1) 锁

事务进行加锁,确保事务进行正确状态 (排他锁 X,共享锁 S)

(2) 用封锁进行并发控制

1) 事务上 X 锁,其他事物必须等 X 锁释放才能进行上锁操作

2) 事务上 S 锁,其他事务可以同时上 S 锁,若上 X 锁,必须等 S 锁释放

3) 读请求上 S 锁, 增删修操作要上 X 锁

(3) 封锁的粒度

封锁的数据单元大小,粒度越细,并发性越高,性能开销越大

通常使用折中粒度

(4) 封锁的级别

通常称 一致性级别 或 隔离度

1) 0 级锁

事务不重写非 0 级别锁更新数据 意义不大

2) 1 级锁

事务不允许重写未成功的数据更新, 防止数据丢失

3) 2 级锁

事务不允许重写也不读取未成功的数据更新, 防止读脏数据

4) 3 级锁 (保证多个事务可串行性)

事务不读取未成功的更新数据,不写任何操作, 防止不可重复读问题

(5) 活锁与死锁

1) 一次性锁请求

事务申请全部的锁请求,全部满足是事务处理才开始(事务串行)

2) 锁请求排序

将每个数据单元标以线性顺序,要求每个事务按此顺序提锁请求  
(排序线性)

3) 序列化处理

设定主任程序,将给定数据单元发给主人,主人单道运行,系统多  
道运行

4) 资源剥夺

每当事务因锁请求不能满足时,强行令一个事务释放锁(注意活  
锁发生)

(6) 可串行性

(7) 两段封锁法

**加锁阶段:** 每个事务发出锁请求,都会获得一个数据单元对象锁,不影  
响其他事务获取其他锁

**释放阶段:** 当事务释放某个对象锁之后,其他事物又可以获得此数据  
单元的对象锁

## 6.5. 备份与恢复

### (1) 导出数据

```
select * from db.table into outfile 'C/backup/table.txt'  
fields terminated by ',' optionally enclosed by '"' line terminated by '?';
```

### (2) 导入数据

```
load data infile 'C/backup/table.txt' into table db.table  
fields terminated by ',' optionally enclosed by '"' line terminated by '?';
```

## 7. 数据库应用设计与开发实例

### 7.1. 需求描述与分析

#### 7.1.1. 功能性需求

#### 7.1.2. 非功能性需求

### 7.2. 系统设计

#### 7.2.1. 功能模块设计

#### 7.2.2. 数据库设计

### 7.3. 系统实现

### 7.4. 系统测试与维护

## 8. 数据管理技术的发展

### 8.1. 数据库技术发展概述

#### 8.1.1. 第一代

1969 年 IBM 公司研制的层次模型 IMS 数据库管理系统

- (1) 支持三级模式(外,模式,内)
- (2) 存储路径来表示数据之间的关系
- (3) 独立的数据定义语言
- (4) 导航的数据操纵语言,嵌入其他高级语言

#### 8.1.2. 第二代

1969 年 IBM 公司 San Jose 研究室研究员 E.F.Codd 发表论文提到数据库关系模型

- (1) 奠定关系模型理论基础
- (2) 研究了关系数据语言(关系代数,关系演算,SQL 等)
- (3) 研制了大量 RDBMS 的原型,攻克了查询优化,并发控制,故障恢复等技术

#### 8.1.3. 新一代

20 世纪 80 年代广泛开展面向对象数据库系统研究 90 年代初获得大量研究成果

- (1) 将支持更丰富的对象结构和规则,集 数据管理,对象管理,知识管理为一体
- (2) 继承第二代的非过程化数据存储方式和数据独立性
- (3) 对其他系统开放
  - 1) 支持数据库语言标准
  - 2) 在网络上支持标准网络协议
  - 3) 系统具有良好的可移植性,可连接性,可扩展性,互操作性等

## 8.2. 数据仓库与数据挖掘

### 8.2.1. 从数据库到数据仓库

#### (1) OLTP 联机事务处理

针对数据日常操作少数据的查询修改

#### (2) OLAP 联机分析处理

对历史数据进行分析,支持管理决策

### 8.2.2. 数据挖掘技术

#### (1) 概念

通过数据挖掘,可以归纳中接触数据的某些特征

#### (2) 关联分析

获取数据库中的一类重要变量,分析其之间存在的规律

#### (3) 分类预测

将规律演变成概念描述来进行模型构造,将数据进行分类和预测

#### (4) 聚类

将获得数据进行分类

#### (5) 孤立点检测

某些数据与全数据存在不一致,特例,也在错误检查和特例分析中 useful

#### (6) 趋势和演变分析

1) 确定业务对象

2) 数据选择

3) 数据预处理

4) 建模,并对模型参数选择合适算法进行优化

5) 模型评估,检查每个步骤是否实现预定目的

6) 模型部署

## 8.3. 大数据管理技术

### 8.3.1. 大数据的定义

(1) 数据量巨大

(2) 数量种类繁多

(3) 处理速度快

(4) 价值密度低

### 8.3.2. 大数据管理技术典型代表

(1) 大数据存储 Hadoop

(2) NoSQL 数据管理系统

1) 文档存储 CouchDB MongoDB

2) 列存储 HBase

3) 图存储

(3) MapReduce 技术