第一章　概述

1.1　数据处理与数据管理

数据处理 指信息的收集, 管理, 加工, 传播等一系列活动的总和

数据管理 数据管理是指对数据进行分类, 组织, 编码, 存储, 检索和维护. 是数据处理的基本环节, 是任何数据处理业务中必不可少的共有部分, 是数据处理的中心问题

1.2　数据库技术

数据库技术主要研究**数据的管理**问题

1.2.1　数据库管理技术的发展

1. 人工管理阶段

2. 文件系统阶段

    1. 数据共享性差, 冗余度大
    2. 程序与数据之间独立性不高
    3. 数据缺乏统一的管理和控制

3. 数据库系统阶段 (**60 年代后期**)

1.2.2 数据库技术的特点

1. **整体结构化 (主要特征, 与文件系统的区别)**, 数据的共享性高, 冗余度小
2. 程序与数据之间的独立性高
3. 数据得到统一的管理和控制

文件系统是数据库技术的基础, 数据库技术中的读写操作通过文件系统实现

1.2.3 开发信息系统时采用数据库技术的原因

尽管有各种各样的信息系统, 但它们的**主要功能都是进行信息处理 (即数据处理)**, 而数据处理都涉及到数据的管理问题. ~~如果数据的管理由各个信息系统自己去实现, 不但耗时耗钱, 而且系统的稳定性, 可靠性, 安全性, 响应时间都不能得到保证.~~ 因为数据管理软件的编写是一项专业性很强的工作, 不是一般的程序员可以胜任的. 采用数据库技术后, 数据的管理就由 DBMS 去完成, 而 DBMS 是专业公司开发的, **系统的稳定性, 可靠性, 安全性, 响应时间都有保证**, 而且**缩短了系统的开发时间, 节约了成本**

1.3 数据库系统 (DBS) 的组成

数据库系统 (DBS) 包括软硬件 *补充? P6*, 由数据库 (DB), 数据库管理系统 (DBMS)(及其开发工具), 数据库应用系统, 数据库管理人员 (DBA) 组成

1.3.1 数据库 (DB)

长期储存在计算机内, 有组织的, 可共享的大量**数据及其联系**的集合

1.3.2 数据库管理系统 (DBMS) 等软件

数据库管理系统 (DBMS) 位于用户与系统间专门负责数据管理的软件, 用户调用 DBMS, **DBMS 调用系统**, 是**数据库系统的核心**

数据库管理系统的 6 大功能

1. 数据定义功能: 数据定义语言 (DDL)

   数据定义语言 (DDL) 主要定义数据库的逻辑结构, 包括定义基本表, 索引和视图三个部分

2. 数据组织, 存储和管理

3. 数据操控功能: 数据操控语言 (DML), **实现数据的增删改查**

   数据操纵语言 (DML) 包括数据查询和数据更新两大类操作, 其中数据更新又包括插入, 删除和修改三种操作

4. 数据库的运行管理和事务管理: 数据控制语言 (DCL)

   数据控制语言 (DCL) 主要有对基本表和视图的授权, 事务控制语句等

5. 数据库的建立和维护功能

1.3.3 数据库管理员 (DBA)

数据库系统中的人员主要有: **数据库管理员** (最重要), 系统分析员各数据库设计人员, 应用程序员, 最终用户

数据库管理员职责

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. 监视数据库的使用和运行
5. 数据库的改进和重组

1.4 数据库的体系结构

1.4.1 三级模式结构

三级模式是对数据的三个抽象级别, 它把数据的具体组织留给 DBMS 管理, 使用户能逻辑地抽象地处理数据, 而不必关心数据在计算机中的具体表示方式和存储方式, 不必考虑存取路径等细节. 另外, 外模式是数据库安全性的一个有力措施, 模式实现了数据的共享, 减少了数据的冗余

1. 外模式

   用户使用的数据视图的描述, 是与某一具体应用有关的数据的逻辑表示. 显然, 外模式是模式的子集, 且可以有多个

2. 模式

   用来描述数据库中全体数据的逻辑结构及特征, 是全体用户数据的最小并集. 数据库模式以某一种数据模型为基础, 综合考虑了所有用户的需求, 并将这些需求有机地结合成一个逻辑整体. 一个数据库只有一个模式

3. 内模式

   数据库中数据的物理结构和存储方法的描述, 是数据在数据库内部的表示方式. 内模式负责定义所有数据的物理存储策略和访问控制方法. 一个数据库只有一个内模式.

1.4.2 二级映像和二级独立性

二级映像保证了数据库系统中的数据具有较高的逻辑独立性和物理独立性

1. 外模式/模式映像, 逻辑独立性

   当模式改变时, 可由数据库管理员改变外模式/模式映像, 使得每个外模式保持不变, 而应用程序是根据外模式编写的, 从而不必修改应用程序

2. 模式/内模式映像, 物理独立性, 数据库系统中**唯**一

   指当内模式改变时, 可由数据库管理员改变模式/内模式映像, 使得模式保持不变 (外模式当然也不变), 从而不必修改应用程序

第二章　关系型数据库

2.1　数据模型

数据模型 对现实世界中某个对象特征的模拟和抽象

2.1.1　数据模型的基本要求

1. 真实地模拟现实世界
2. 容易被人理解
3. 便于在计算机上实现

2.1.2　数据模型的三个层次

1. 概念模型
2. 逻辑模型
3. 物理模型

2.1.3　数据模型的三个组成要素

1. 数据结构
2. 数据操作
3. 完整性约束条件

2.2 概念模型

概念模型 也称信息模型, 是**按用户的观点对数据和信息建模**, 独立语计算机系统的模型

概念模型具有较强的语义表达能力, 能够方便, 直接地表达应用所涉及到的现实世界中的各种语义知识, 另一方面它概念简单, 清晰, 易于用户理解, 且不依赖于具体的计算机系统。概念模型主要用于数据库设计, 是用户与数据库设计人员之间进行交流的桥梁

2.2.1 基本概念

1. 实体
2. 属性
3. 码

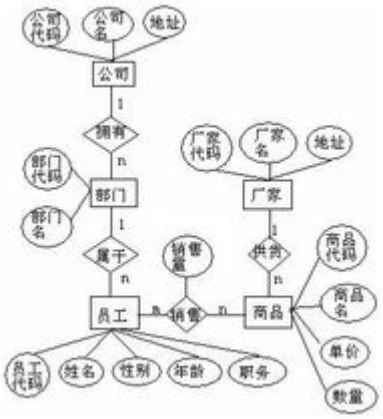
    1. 主码 $\in$ {候选码 $\subseteq$ 超码}
    2. 主属性 $=$ {属性 $\in$ 候选码}
    3. 全码: 所有属性组是候选码

4. 域
5. 实体型
6. 实体集

2.2.2 实体间的联系

1. 一对一
2. 一对多
3. 多对多

2.2.3 概念模型的表示方法: ER 图



绘制步骤

1. 确定实体
2. 确定各实体的属性
3. 确定实体之间的联系
4. 确定各联系的属性

2.3 逻辑模型

逻辑模型有层次, 网状, 关系 (**最广泛**), 面向对象, 对象关系数据模型 5 种

2.3.1 非关系模型的优缺点

优点: 查询效率高

缺点: 数据结构复杂, 不易理解, 编程也复杂

2.3.2 关系模型

数据结构是一张二维表, 其优点是:

1. 有严格的理论基础, 概念单一

   在关系模型中的概念单一主要体现在数据结构单一 (即现实世界中实体以及实体之间的各种联系都用关系来表示), 而数据结构单一又带来操作符的统一。操作符的统一体现在无论是插入 (或删除修改) 一个实体值还是插入 (或删除修改) 一个联系值, 操作的命令是相同的 (不像 DBTG 中用 STORE 插入实体值, 用 CONNECT 插入联系值), 这大大方便了用户的使用

2. 数据结构简单, 清晰, 用户易于理解

   所谓存储路径对用户透明就是指用户不需要知道数据的物理结构和存储方式. 优点: 因为存取路径对用户透明, 从而具有更高的数据独立性, 更好的安全保密性, 简化了程序员的工作. 缺点: 因为存取路径对用户透明, 导致查询效率不如非关系系统, 增加了开发 DBMS 的难度

3. 存取路径对用户透明, 程序与数据的独立性高, 易于应用程序的编写和维护

缺点:

1. 查询效率不如非关系模型, DBMS 的开发难度高

关系模型的数据结构

分量 列 (不可再分)
元组 行

关系模型的完整性

1. 实体完整性: 主码中的属性不能取空值

2. 参照完整性
3. 用户定义完整性

RDBMS 的完整性控制机制都应有完整性定义, 完整性检查和违约处理这三方面的功能

关系操作的分类

最常用的关系操作有:

查询操作 选择, 投影, 连接, 除法, 并, 交, 差, 广义笛卡儿积
更新操作 增加, 删除, 修改

关系操作的特点

1. 关系操作采用集合操作方式
2. 高度的非过程化

2.4 关系代数

非过程化语言 用户只要告诉系统操作的要求, 不必告诉系统如何来完成该操作 (即用户只要告诉系统 “做什么”, 而不必告诉 “怎么做”) 的语言就是非过程化语言

2.4.1 传统集合运算符

1. **并** $\cup$
2. **差** $-$
3. 交 $\cap$
4. **广义笛卡儿积** $\times$

并, 差, 广义笛卡儿积都是基本运算

交是非基本运算

2.4.2 专门的关系运算符

1. **选择** (`SELECT * WHERE F`) $\sigma_F(R)$ (F 为判断条件)
2. **投影** (`SELECT A, B, ...`) $\pi_{A[,B...]}(R)$
3. 连接 $R \underset{R.A_0 \Theta S.A_1}{\bowtie} S, \Theta \in \{>, <, =, \ldots\}$
   等值连接: $\Theta \to =$ 自然连接: 被连接的两个关系中进行相等比较的分量必须是相同的属性 (组), 并且要在结果中把重复的属性 (组) 去掉
4. 除 $R \div S$

选择, 投影是基本运算

连接, 除是得基本运算

2.4.3 扩展的关系运算符

1. 外连接 $R \bowtie S$ (= 在哪边保留哪边元组)
2. 广义投影 (`SELECT A, B, F(C), ...`) $\pi_{A[,B[,F(C)...]]}(R)$
3. 聚合 (`SELECT F1, F2, ... GROUP BY A, B, ...`) ${}_{A,B,...}\mathcal{G}_{F_1,F_2,...}(R)$:
   指定若干个属性组成新的关系

第三章　SQL

3.1　T-SQL for school

```
CREATE DATABASE database_name ON (
    name = database_name, filename = 'file_path',
    size = 5, filegrowth = 2
)

CREATE TABLE table_name (
    column_name_1 INT
        FOREIGN KEY REFERENCES other_table(column)
            /* ON INSERT 不存在, 默认 NO ACTION */
            ON DELETE CASCADE /* NO ACTION | SET NULL | SET DEFAULT */
            ON UPDATE NO ACTION,
    column_name_2 SMALLINT UNIQUE,
    column_name_3 DECIMAL(12, 1) CHECK (column_name_3 > 0),
    column_name_4 CHAR(5) NOT NULL PRIMARY KEY,
    column_name_5 VARCHAR(5),

    CONSTRAINT constraint_name_a
        CHECK (column_name_3 > column_name_1 AND ...)
    CONSTRAINT constraint_name_b
        PRIMARY KEY (column_name_1, column_name_2)
```

```
)

CREATE /* [UNIQUE] [CLUSTERED | NONCLUSTERED] */ INDEX index_name
ON table_name(column_1, column_1) /* ASC | DESC */
DROP INDEX index_name

CREATE VIEW view_name /* (column_name, ...) */ AS
SELECT ...

ALTER TABLE table_name ADD column_name DATE NOT NULL
ALTER TABLE table_name DROP COLUMN column_name
ALTER TABLE table_name ALTER COLUMN column_name INT

ALTER TABLE table_name
    ADD CONSTRAINT constraint_name PRIMARY KEY (primary_key, ...)
ALTER TABLE table_name DROP CONSTRAINT constraint_name

DROP TABLE table_name

SELECT /* TOP 1 */ DISTINCT
    column, COUNT(DISTINCT column), SUM(column),
    AVG(column), MAX(column), MIN(column) AS A
FROM table_name, view_name, ... /* 笛卡儿积 */
/* FROM table_name
   [INNER | {LEFT | RIGHT | FULL} [OUTER]] JOIN
   table_name ON ... */
WHERE
    column_1 NOT IN (data, ...)
    AND column_2 LIKE '_A%' /* _ -> ?, % -> * */
    OR column_3 BETWEEN a AND b
    AND column_4 IS NOT NULL
```

```
    OR column_5 EXIST /* > {{ANY | SOME} | ALL} */ (SELECT ...)
GROUP BY column_name, ...
HAVING logical_expression
ORDER BY column_name DESC
UNION /* INTERSECT | EXCEPT */
SELECT ...

INSERT INTO table_name VALUES (column_data, ...)
INSERT INTO table_name (column_name_1, ...) VALUES (column_data, ...)

DELETE FROM table_name WHERE logical_expression
UPDATE table_name SET column_name = data WHERE logical_expression

CREATE LOGIN login_name WITH PASSWORD = 'password'
DROP LOGIN login_name

/* in database */
CREATE USER user_name FOR LOGIN login_name
DROP USER user_name

GRANT SELECT, UPDATE(column_name, ...)
ON table_name TO user_name WITH GRANT OPTION
REVOKE /* ALL */ SELECT, INSERT, DELETE, UPDATE, REFERENCES
ON table_name FROM u_many CASCADE
```

3.2 SQL 语言的主要特点

1. 综合统一, 即集 DDL, DML 和 DCL 功能于一体
2. 面向集合的操作方式, 即操作的对象和操作的结果都是元组的集合
3. 高度非过程化, 即在完成某项查询要求时, 用户无需了解存取路径, 只要提出 “做什么”, 不必指出 “怎么做”

4. 以同一种语法结构提供两种使用方式, 即独立地用于联机交互的使用方式和嵌入到高级语言中这两种不同的使用方式下, 语法结构是基本上一致的
5. 支持三级模式结构

第四章　索引和视图

4.1　索引

4.1.1　索引分类

稀疏索引和稠密索引

聚集索引和辅助索引 (储存结构)

聚集索引 聚集索引就是指表中的元组按照索引中搜索码指定的顺序排序, 使得具有相同搜索码值的元组在物理上聚集在一起. 显然, 一张表最多只能有一个聚集索引. 聚集索引往往是稀疏索引, 可以只存储部分搜索码值

辅助索引 因为表中的元组是按聚集索引而不是辅助索引的搜索码有序存放的, 所以辅助索引必须是稠密索引, 对每个搜索码值都有一个索引项, 包含指向表中每个元组的指针. 显然, 一张表可以创建多个辅助索引

聚集索引应在辅助索引前建立

唯一索引和非唯一索引 (搜索值是否允许重复)

单索引和复合索引 (搜索码中的属性数)

4.2 视图

基本表 基本表在数据库中既要存放它的定义, 又要存放它的数据

视图 视图是从一张或几张基本表 (或视图) 导出的表. 与基本表不同, 视图是一张虚表, 在数据库中只存放视图的定义 (即 SELECT 语句), 不存放视图对应的数据 (即 SELECT 语句的查询结果)

两者的联系是视图的数据在它对应的基本表中, 所以基本表中的数据一旦发生变化, 从视图中查询出的数据也就随之改变

4.2.1 视图更新的基本原则

1. FROM 子句中只有一个数据库关系
2. SELECT 子句中只包含关系的属性名, 不包含任何表达式、聚集函数或 DISTINCT 短语
3. 没有出现在 SELECT 子句中的属性可以取空值, 也不是主码中的属性
4. 子查询中没有 GROUP BY 或 HAVING 子句

第五章　数据库安全技术

数据库的安全性 是指保护数据库以防止不合法的使用所造成的数据泄漏, 更改或破坏.

对比数据库完整 数据库的完整性是指数据库的任何状态变化都能反映真实存在的客观世界的合理状态，数据库中的数据应始终保持正确且合理的状态.

也就是说，数据库的完整性是指数据的正确性和相容性. 两者的定义已经看出两者的区别，而两者的联系主要体现在触发器机制既可以用来实现完整性，也可以用来实现安全性

5.1　安全等级

TCSEC/TDI 将系统划分为 DCBA 4 组, D, C1, **C2** (安全产品的最低档次), B1, B2, B3 和 A1

5.2　数据库安全性控制的常用方法和技术

1. 用户标识和鉴别
2. 存取控制 (授权机制)
3. 视图机制
4. 审计
5. 数据加密

第六章　事务管理

所谓事务是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

6.1　事务的 ACID 性质

1. 原子性: 事务中包含的数据库操作要么都做, 要么都不做。这一性质即使在系统发生各种故障之后仍能得到保证
2. 一致性 (正确性): 保证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
3. 隔离性: 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, 即每个事务都感觉不到系统中有其他事务在并发地执行
4. 持续性: 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

6.2　并发控制

并发执行的好处 提高吞吐量和资源利用率

6.2.1　并发操作与数据的不一致性

破坏事务的隔离性

1. 丢失修改

T1	T2
读 A = 20	
	读 A = 20
A = A - 1	
写回 A = 19	
	A = A - 1
	写回 A = 19

2. 读 '脏' 数据

T1	T2
读 A = 50	
A = A - 30	
写回 A = 20	
	读 A = 20
ROLLBACK	
A 恢复 50	

3. 不可重复读

T1	T2
读 A = 30	
读 B = 50	
求和 = 80	
	读 B = 50
	读 B = B - 30
	写回 B = 20
读 A = 30	
读 B = 20	

T1	T2
求和 = 50	

4. (幻影现象), 后两者

    1. 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, 发现某些记录消失了
    2. 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

6.2.2 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顺序串行地执行这些事务时的结果相同, 称这种并发调度为可串行化调度. 一个给定的并发调度, 当且仅当它是可串行化的, 才认为是正确调度

6.2.3 封锁及封锁协议

封锁 事务 T 在对某个数据对象 (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

封锁类型

排他锁 (X 锁)(写锁), 共享锁 (S 锁)(读锁)

封锁协议

1. 一级封锁协议 (防止丢失修改): 在修改数据 R 前必须加 X 锁, 直到事务结束释放

2. 二级封锁协议 (防止读 '脏' 数据): 一级 + 读 R 前加 S 锁, 读完放
3. 三级封锁协议 (防止不可重复读): 一级 + 读 R 前加 S 锁, 事务结束放

死锁

死锁就是两个或多个事务竞争对相同资源的控制权而相互无限制的等待. 预防死锁的方法通常有一次封锁法和顺序封锁法两种. 诊断死锁的方法通常有超时法和事务等待图法两种. 并发控制机制检测到系统中存在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, 将其撤消, 释放此事务持有的所有的锁, 使其它事务能继续运行下去

6.2.4 两端锁协议

- 在对任何数据进行读, 写操作之前, 事务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

6.2.5 锁的粒度

封锁粒度与系统的并发度和并发控制的开销密切相关. 封锁的粒度越大, 数据库所能够封锁的数据单元就越少, 并发度就越小, 系统开销也越小; 反之, 封锁的粒度越小, 并发度较高, 但系统开销也就越大. 如果在一个系统中同时支持多种封锁粒度供不同的事务选择是比较理想的, 这种封锁方法称为多粒度封锁. 通过选择适当的封锁粒度来平衡封锁开销和并发度两个因素, 以求得最优的效果

意向锁

意向锁的含义是如果对一个结点加意向锁, 则说明该结点的下层结点正在被加锁; 对任一结点加锁时, 必须先对它的上层结点加意向锁. 如果没有意向锁, 那么系统在对某个数据对象加锁时, 系统要检查该数据对象上有

无显式封锁与之冲突; 再检查其所有上级结点, 看本事务的显式封锁是否与该数据对象上的隐式封锁冲突; 还要检查其所有下级结点, 看它们的显式封锁是否与本事务的隐式封锁冲突. 显然, 这样的检查方法效率很低. 引进意向锁后, DBMS 就无需逐个检查下级结点的显式封锁, 提高了对某个数据对象加锁时系统的检查效率

1. IS 锁: 如果对一个数据对象加 IS 锁, 表示它的后裔结点拟 (意向) 加 S 锁
2. IX 锁: 如果对一个数据对象加 IX 锁, 表示它的后裔结点拟 (意向) 加 X 锁
3. SIX 锁: 如果对一个数据对象加 SIX 锁, 表示对它加 S 锁, 再加 IX 锁, 即 SIX = S + IX

6.3 数据库恢复

6.4 备份技术

6.4.1 数据库备份

静态转储方法 优点是转储得到的一定是一个数据一致性的副本, 缺点是转储期间不允许有对数据库的任何存取或修改活动

动态转储方法 优缺点正好与静态转储相反, 也建议在系统不繁忙时备份

海量转储方法 优点是得到的后备副本进行恢复往往更方便, 缺点是耗时耗存储空间

增量转储方法 优缺点正好与海量转储相反

6.4.2 登录日志备份

日志文件是用来记录事务对数据库的更新操作的文件. 如果没有日志文件就不能将数据库恢复到动态转储结束时刻的一致性状态, 更不能恢复到故障发生前某一时刻的一致性状态. 有了日志文件后, 首先由 DBA 重装

数据库后备副本, 将数据库恢复至转储结束时刻的状态 (数据库未被破坏这一步不要做), 然后利用日志把自转储结束时刻 ~ 故障发生时刻所有已完成的事务进行重做处理, 所有未完成的事务进行撤销处理, 即可把数据库恢复到故障发生前某一时刻的正确状态

登录日志文件的基本原则

1. 登录的次序严格按并行事务执行的事件次序

2. 必须先写日志文件, 后写数据库

   原因: 防止无记录操作

6.5 恢复技术

6.5.1 事务故障恢复

事务故障不破坏数据库, 但破坏了事务的原子性, 造成了数据的不一致性

事务故障的恢复策略是对发生事务故障的事务进行撤销 (UNDO) 处理

6.5.2 系统故障恢复

系统故障不破坏数据库, 但破坏了事务的原子性和持续性, 造成了数据的不一致性

系统故障的恢复策略是强行撤销所有未完成的事务, 让所有非正常终止的事务回滚, 清除它们对数据库的所有修改; 同时还需要重做 (REDO) 所有已提交的事务, 将已完成事务提交的结果写入数据库, 将数据库恢复到一致状态

6.5.3 介质故障恢复

介质故障破坏了数据库本身, 也破坏了事务的原子性和持续性 o

事务故障的恢复策略是对发生事务故障的事务进行撤销 (UNDO) 处理. 系统故障的恢复策略是强行撤销所有未完成的事务, 让所有非正常终止的事务回滚, 清除它们对数据库的所有修改; 同时还需要重做 (REDO) 所有已提交的事务, 将已完成事务提交的结果写入数据库, 将数据库恢复到一致状态. 介质故障的恢复策略是重装数据库, 再重做已完成的事务

发生故障时, 一方面一些尚未完成的事务对数据库的更新可能已写入磁盘上的物理数据库, 另一方面有些已完成的事务对数据库的更新可能有一部分甚至全部还留在缓冲区, 尚未写入磁盘上的物理数据库, 故障使得缓冲区中的内容都被丢失. 所以对没有提交的事务要 UNDO, 对提交的事务要 REDO

6.6 检查点恢复技术

利用日志技术进行数据库恢复时, 恢复子系统需要检查所有日志记录, 确定哪些事务要重做, 哪些事务要撤销. 这样做会带来两个问题, 一是搜索整个日志将花费大量的时间, 二是很多需要重做处理的事务实际上已经将它们的更新操作结果写到了数据库中, 然而恢复子系统又重新执行了这些操作, 浪费了大量时间. 使用检查点方法可以改善恢复效率, 因为该方法保证在检查点之前提交的事务对数据库的修改一定都已写入数据库, 进行恢复处理时, 没有必要对这些事务做重做操作

第七章　关系型数据库设计理论

7.1　函数依赖

平凡函数依赖 $(x, y) \to x$
完全函数依赖 $X \xrightarrow{f} Y$
部分函数依赖 $X \xrightarrow{r} Y$
传递函数依赖 $X \xrightarrow{p} Y$

7.1.1　闭包推导

补充？*P184*

7.1.2　候选码推导

补充？*P185*

7.2　关系模式规范化

目的 在关系数据库中, 对关系模式的最低要求是满足第一范式。但满足最低要求的关系模式往往不能很好地描述现实世界, 可能会存在插入异常、删除异常、修改复杂以及数据冗余等问题, 需要寻求解决这些问题的方法

基本思想 逐步消除数据依赖中不合适的部分, 使各关系模式达到某种程度的 '分离'.

实现 通过模式分解把一个低级别范式的关系模式逐步转换为若干个高级别范式的关系模式集合来完成的

实质 概念的单一化。

1. 1NF
2. 2NF
3. 3NF
4. BCNF

7.3 关系模式分解

二者独立

1. 无损连接性: 保证信息不丢失

   $U_1 \cap U_2 \rightarrow U_1 - U_2 \in F^+ or U_1 \cap U_2 \rightarrow U_2 - U_1 \in F^+$

2. 保持函数依赖: 在减轻或解决数据冗余和各种操作异常情况的同时, 保证分解后的关系模式保留了原关系模式的所有语义

   $F^+ = G^+$

第八章　数据库设计理论

8.1　数据库设计过程

1. 需求收集与分析, 就系统功能、数据需求、数据完整性和安全性等诸多方面与应用领域专家和用户展开多种方式的沟通, 同时综合不同用户的应用需求
2. 概念结构设计, 将用户需求以概念模型的方式表达出来, 用以和用户沟通并确认需求
3. 逻辑结构设计, 将概念模型转换成具体的数据库产品支持的逻辑模型, 形成数据库的逻辑模式并对它进行优化并形成数据的外模式
4. 物理结构设计, 为逻辑模型选取一个最合适应用环境的物理结构, 形成数据库内模式
5. 最后实施、运行和维护, 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

索引