

佛脚 >  数据管理基础

 编辑 

# 90-期末复习

## 题型

- 单选 2%\*15
- 简答 8%\*5
- 问答 10%\*3
- 作业都需要会

## 背诵

以下为需要完整背诵的概念

## 关系型数据库

### 人工、文件、数据库的场景

- 人工：“挥发性”计算，无需持久化保存，无共享，冗余度大，无结构
- 文件：“持久性”计算，共享性差，冗余度大，记录内有结构，整体无结构，独立性差
- 数据库：“共享性”计算

## 三个模型

类别	概述	应用场景
概念模型	按用户观点对数据和信息建模	数据库设计
逻辑模型	按计算机系统观点对数据和信息建模	DBMS实现
物理模型	描述数据在系统内部的表示和存取方法	数据库实现

## 三级模式、两重映射

- 三级模式
  - 外模式：数据库用户使用的局部数据的逻辑结构和特征描述
  - 模式：数据库中全体数据的逻辑结构和特征描述
  - 内模式：数据物理结构和存储方式的描述
- 两重映射
  - 外模式/模式映像：定义外模式和模式的对应关系，保证数据逻辑独立性
  - 模式/内模式映像：定义模式和内模式的对应关系，保证数据物理独立性

## 安全

### 自主存取控制

- 用户对不同对象拥有不同的存取权限
- 不同用户对同对象的存取权限不同
- 用户可以将权限授予其他用户或收回权限
  - `GRANT privilege ON sth TO sb`
  - `REVOKE privilege ON sth FROM sb`
  - 为所有用户授权/收回时，使用 `PUBLIC`
- 缺点：数据本身无安全性标记，可能存在数据的“无意泄露”

### 强制存取控制

- 数据对象被标以密级
- 用户被授予许可证
- 只有拥有合法许可证的用户才能访问数据
- 用户无法直接感知/进行控制
- 实体分类
  - 主体：系统中的活动实体，实际用户、进程
  - 客体：系统中的被动实体，文件、基本表、索引、视图
- 控制规则
  - 读：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（低级别的用户不能读高级别的对象，以防高级别数据泄漏）
  - 写：仅当主体的许可证级别小于或等于客体的密级时，该主体才能写相应的客体（高级别的用户不能写低级别的对象，以防高级别数据泄漏）

## 完整性

### 完整性约束条件

完整性类型	定义	SQL 语法示例	选课系统中的例子
实体完整性	主码属性不能为空；每个元组必须唯一标识	PRIMARY KEY	表 Student (Student ID, Name, ...) 中, StudentID 是主码, 必须唯一且非空
参照完整性	外码必须引用另一个表中存在的主码或为空	FOREIGN KEY (...) REFERENCES ...(...)	表 Enrollment (StudentID, CourseID, ...) 中, StudentID 引用 Student (Student ID)
用户定义完整性	业务逻辑约束, 由用户根据需求自定义	CHECK、 UNIQUE、 NOT NULL、 CONSTRAINT	Grade 表中, Score 必须在 0~100 之间: CHECK (Score BETWEEN 0 AND 100)

## 数据库设计

### 垂直/水平分解

- 水平分解：把关系的元组分为若干子集，定义每个子集为一个子关系（按照行拆分表）
  - 使用场景：部分元组较为常用，而其他元组不常用
- 垂直分解：把关系的属性分为若干子集，定义每个子集为一个子关系（按照列拆分表）
  - 优点：提升某些事务效率
  - 缺点：部分事务需要连接操作，降低效率
  - 适用场景：某些属性经常一起使用，且其他属性不经常使用，取决于分解后的总效率是否提高

### 聚簇

- 原理
  - 独立聚簇：将某属性的值相同的元组集中存放在连续的物理块中，减少磁盘 I/O
  - 组合聚簇：把多个连接的元组按连接属性值聚集存放，加速连接查询
- 目的：提高 `ORDER BY`、`GROUP BY`、`UNION`、`DISTINCT`、连接的性能
- 适用场景：教务系统中，学生院系表按照院系聚簇存储，方便查询同一院系的学生信息
- 不适用场景：教务系统中，学生表按照籍贯聚簇，籍贯查询不频繁，聚簇会降低查询效率

## 索引

- 定义：用于加速数据检索的数据结构，它在数据库表的一个或多个列上创建，用于快速定位满足查询条件的记录，适用于经常用来查询/连接的属性
- 实现方式：B树、B+树、哈希索引、位图索引

```
CREATE [UNIQUE] [CLUSTER] INDEX <index_name> ON <table_name>
(<column_name> [ASC | DESC], ...)
```

## 事务

### 事务的特性

- 原子性 Atomicity：要么全做，要么全不做
- 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（完整性约束没有被破坏）
- 隔离性 Isolation：并发执行的事务之间互不干扰
- 持久性 Durability：事务一旦提交，对数据库的修改是永久性的

## 转储

- 转储：定期将数据库完整复制到其他介质
- 恢复到故障发生时的状态需重新运行转储以后的所有更新事务

## 日志

- 记录事务对数据库的更新操作的文件
- 用途：事务故障、系统故障、介质故障恢复
- 登记日志原则
  - 登记次序需严格按照事务的执行顺序
  - 先登记后执行（多一次无效的撤销不会影响数据正确性）
- 恢复方式：重做已完成的事务，撤销未完成的事务

## 检查点

- 检查点：在日志文件中增加检查点、重新开始文件，动态维护日志
  - 检查点记录：建立时刻正在执行的事务清单，事务最近一个日志记录的地址
  - 重新开始文件：“检查点记录”在日志文件中的地址
- 维护日志文件步骤：周期性建立检查点，保存数据库状态
  1. 将日志缓冲区中的日志写入日志文件
  2. 在日志文件中增加检查点记录
  3. 将数据缓冲区的数据记录写入数据库
  4. 将检查点记录在日志文件中的地址写入重新开始文件
- 恢复步骤
  1. 从重新开始文件中找到最后一个检查点记录在日志文件中的地址，由此找到最后一个检查点记录
  2. 将所有正在执行的事务放入 **ACTIVE** 队列，将 **ACTIVE** 队列中的事务放入 **UNDO** 队列
  3. 从检查点记录开始，向后扫描日志文件
    - 若有新事务：加入 **UNDO** 队列
    - 若有已提交事务：移入 **REDO** 队列
  4. **UNDO** 队列中的事务进行撤销，**REDO** 队列中的事务进行重做

## 并发控制

## 不一致性

- 丢失修改：A 写后 B 写，导致 A 写的内容被覆盖
- 不可重复读：A 读后 B 写后 A 读，当 A 两次读取的数据不一致（可能为增/改/删）
- 读脏数据：A 写后 B 读后 A 撤销写，B 读取了撤销前的数据，产生错误

## 锁的种类

类型	别称	上锁者权限	他人权限	作用
X锁	排他锁/写锁	独占，可读可写	无	写者为避免他人读取错误数据
S锁	共享锁/读锁	可读	可读（只能上 S 锁，不能上 X 锁）	读者为避免他人修改数据

- 步骤题中语法：`Slock X`、`Xlock X`、`Unlock X`

## 三种封锁协议

类型	写	读	丢失修改	读脏数据	不可重复读
一级封锁协议	上 X 锁，事务结束释放	不上锁	Y	N	N
二级封锁协议	上 X 锁，事务结束释放	上 S 锁，读取后释放	Y	Y	N
三级封锁协议	上 X 锁，事务结束释放	上 S 锁，事务结束释放	Y	Y	Y

## 死锁活锁

- 活锁：系统先满足后来事务的请求，先来的事务一直在等待锁，无法继续执行
  - 采用先来先服务的方式

- 死锁：两个或多个事务互相等待对方释放锁，导致无法继续执行

## 解决死锁

- 一次封锁法：事务将所需的数据一次性上锁
- 顺序封锁法：定义上锁顺序，所有事务的上锁顺序一致
- 超时法：事务等待锁的时间超过一定阈值则放弃
- 等待图法：间歇性生成事务等待图，如果存在回路则死锁

## 两段锁

- 实现步骤
  - 扩展阶段：在需要用数据前，先上锁
  - 收缩阶段：释放锁，开始释放锁后不能再上锁
- 两段锁协议 → 可串行化（仅为充分条件）
  - 不满足两段锁协议的调度不一定不可串行化
- 和一次封锁法的差别：
  - 一次封锁法必须一次性对所有所需数据上锁
  - 两端锁协议更宽松，可上锁后对数据进行操作，随后继续上锁，只需保证上锁和释放过程不交叉
  - 因此两段锁协议的事务可能会发生死锁

## 意向锁

- 封锁力度：上锁对象的大小
- 多粒度封锁：根据事务所需数据的粒度，动态调整封锁粒度，父节点上锁时，子节点自动上锁
- 目的：提高加锁时系统的检查效率
- 意向锁：当子节点加基本锁时，须对所有父节点加意向锁
  - IS（意向共享锁）：子节点上 S 锁时，父节点上 IS 锁
  - IX（意向排他锁）：子节点上 X 锁时，父节点上 IX 锁
  - SIX（共享意向排他锁）：同时对对象上 S 锁和 IX 锁

T1\T2	S	X	IS	IX	SIX
S	Y	N	Y	N	N
X	N	N	N	N	N
IS	Y	N	Y	Y	Y
IX	N	N	Y	Y	N
SIX	N	N	Y	N	N
-	Y	Y	Y	Y	Y

- 具有意向锁的多粒度封锁方法
  - 申请封锁：从上向下对数据库加意向锁，检查是否存在不相容的锁
  - 释放封锁：从下向上
  - 好处：无需检查对象的子对象的锁状态，减少开销

## 理解

以下内容多出现在选择/大题中，无需背诵，仅需理解即可

## 关系的基本概念

- 关系：笛卡尔积的子集/二维表
- 元组：关系中的元素/表中的一行
- 属性：表中的一列
- 码
  - 候选码：唯一标识元组的属性集，且其任意真子集都不能唯一标识元组
  - 主码：选定多个候选码中的一个
  - 主属性：主码中的属性
  - 外码：表中的属性，引用另一表的主码
- 关系模式：关系的描述，关系名(属性1, 属性2, ...)

- 关系模型必须是规范化的，每一个分量是不可分的数据项，属性不能再分

## 基本关系的性质

- 列是同质的：每一列中的分量来自同一个域
- 不同的列可以出自同一个域：不同列的定义域可能相同
- 行、列的顺序无关紧要
- 任何两个元组的候选码不能相同
- 分量必须取原子值：每一个分量都是不可再分的数据项

## 关系运算

- 基本运算符：选择、投影、并、减、笛卡尔积
- 选择（行的运算）： $\sigma_F(R) = \{t|t \in R \wedge F(t)\}$ ,  $F$  是布尔表达式（返回真或假），筛选满足  $F$  的元组
- 投影（列的运算）： $\Pi_A(R) = \{t[A]|t \in R\}$ ,  $A$  是属性名，筛选  $A$  对应的列
  - 可能会导致行的减少：列删除后可能会产生重复的元组
- 连接： $R \underset{A \theta B}{\bowtie} S = \{\widehat{t_r t_s} | t_r \in R \wedge t_s \in S \wedge (t_r[A] \theta t_s[B])\}$ 
  - 在笛卡尔积后，选择符合条件，即  $t_r[A] \theta t_s[B]$  为真的元组
  - 等值连接： $\theta$  是等于号，比较两个属性的值是否相等
  - 自然连接：特殊的等值连接， $R$  和  $S$  中有相同的属性，只保留一次（在行运算的基础上去除重复列）
  - 悬浮元组 Dangling tuple：在自然连接中， $R$  中某些元组有可能在  $S$  中不存在公共属性上值相等的元组，导致  $R$  中的元组被舍弃
  - 外连接：保留悬浮元组，空值填充 NULL
  - 左/右外连接：只保留左/右表的悬浮元组，悬浮元组中右/左表独有的属性置为 NULL
- 除（行和列的运算）： $R \div S = \{t_r[X] | t_r \in R \wedge \Pi_Y(S) \subseteq Y_X\}$ ,  $A$  和  $B$  是属性名
  - $R$  有属性  $X$  和  $Y$ ,  $S$  有属性  $Y, Z$ ,  $Y_R$  和  $Y_S$  的域相同

- 找出满足所有 $Y$ 的 $X$  ( $X$ 的象集能够包含 $\pi_Y(S)$ )
- $x$ 在 $R$ 中的象集:  $Y_x = \{t[Y] | \exists t \in R, t[X] = x\}$

① 写出查询表达式的题先投影、选择再连接

## ER图的绘制

- 实体型: 矩形
- 属性: 椭圆形
- 联系: 菱形
  - 通过无向边连接实体和联系, 写上联系类型
  - 联系可以具有属性

## 逻辑结构设计的转换原则

- 实体型 → 关系模式
  - 实体的属性 → 关系的属性
  - 实体的码 → 关系的码
- 1:1 联系
  - → 独立的关系模式 (新开一张表)
  - 和某一端实体对应的关系模式合并 (作为某一端的属性)
- 1:n 联系
  - → 独立的关系模式
  - → 和n端的对应关系模式合并
- m:n 联系 → 独立的关系模式
- 多元联系 (三个及以上实体间的联系) → 独立的关系模式
- 具有相同码的关系模式可合并
- **每个实体需加入编号作为主键**
- 需根据题意加入合适的信息: 类别、数量.....

# 关系范式

## 数据依赖

- 函数依赖:  $X \rightarrow Y$ , 属性集  $X$  可以唯一确定  $Y$
- 完全函数依赖:  $Y$  依赖于  $X$  的所有属性, 任意  $X$  的真子集都不能唯一确定  $Y$
- 传递依赖:  $X \rightarrow Y, Y \rightarrow Z, Y \not\rightarrow X, Z \not\subseteq Y$ , 记作  $X \xrightarrow{\text{传递}} Z$ 
  - 若  $X \rightarrow Y, Y \rightarrow Z, Y \rightarrow X$ , 则  $Z$  直接依赖于  $X$ , 不算传递函数依赖

## 定义

- 1NF: 每个分量都是不可分的原子值, 不能出现多个项  $Y = \{a_1, a_2\}$  的情况
- 2NF: 非主属性必须完全依赖于候选码, 消除部分依赖
- 3NF: 非主属性必须直接依赖于候选码, 消除传递依赖
- BCNF
  - 所有非主属性都完全函数依赖于每个候选码
  - 所有主属性都完全函数依赖于每个不包含它的候选码
  - 没有任何属性完全函数依赖于非码的任何一组属性

## 提高范式

- 1NF  $\rightarrow$  2NF: 把部分依赖的候选码和属性拆到新的表内
- 2NF  $\rightarrow$  3NF: 把传递依赖的候选码和属性拆到新的表内
- 3NF  $\rightarrow$  BCNF: 找出不满足 BCNF 的候选码和属性拆到新的表内

三  南软佛脚玩乐指南

Github



判断流程

- 候选码：唯一标识元组的属性集，且其任意真子集都不能唯一标识元组
- 主码：选定多个候选码中的一个
- 主属性：主码中的属性
- 非主属性：不属于任何候选码的属性
- 超码：候选码的超集

1. 求候选码、主属性、非主属性
2. 所有函数依赖的左边都是超码  $\rightarrow$  BCNF
3. 左边非超码的函数依赖的右边是主属性  $\rightarrow$  3NF
4. 任意候选码的真子集都无法推出非主属性  $\rightarrow$  2NF
5. 所有属性都是原子值  $\rightarrow$  1NF

## 事务调度

- 可串行化调度：多事务并发后的结果和事务按某一顺序串行执行的结果一致
  - 并发调度当且仅当其为可串行化时才是正确调度
- 冲突可串行化：若调度  $Sc$  在保证冲突操作次序不变的情况下，交换事务不冲突操作的次序，得到串行的调度  $Sc'$ ，则  $Sc$  是冲突可串行化的
  - 不能交换的操作：
    - 同一事务的两个操作
    - 不同事务的冲突操作
  - 冲突可串行化  $\rightarrow$  可串行化

## SQL

### 创建表格

- 列约束：
  - `NOT NULL`：非空约束
  - `UNIQUE`：唯一约束
  - `PRIMARY KEY`：主键约束

- 表约束:

- PRIMARY KEY (<column\_name>, [<column\_name>]) : 主键约束, 可以包含多个列
- FOREIGN KEY (<column\_name>) REFERENCES <table\_name> (<column\_name>) : 外键约束

```
CREATE TABLE Course (
    Cno INT PRIMARY KEY,
    Cname CHAR(40) NOT NULL,
    Ccredit SMALLINT,
    FOREIGN KEY (Cpno) REFERENCES Course(Cno)
);
```

## 查询数据

- SELECT** 要显示的列
  - 目标列表达式可以是表达式/函数, 如 <column\_name> + 1, LOWER(<column\_name>)
  - 可以使用 AS 给列起别名, 也可以不用 AS, 如 SELECT <column\_name> AS <alias\_name>, SELECT <column\_name> <alias\_name>
  - DISTINCT : 去重
  - ALL : 保留重复值 (默认)
  - \* : 所有列
- FROM** 要查询的表/视图
- WHERE** 查询条件
  - 比较运算符: =, <>, !=, <, <=, >, >=, !<, !>, NOT + 运算符
  - 范围: BETWEEN <low> AND <high>, NOT BETWEEN <low> AND <high>
  - 包含: IN (<value>, ...), NOT IN (<value>, ...)
  - 字符匹配
    - LIKE : 模糊匹配, % 表示0个或多个任意字符, \_ 表示单个字符
    - NOT LIKE
    - 指定转义符: ESCAPE <escape\_char>, 如 LIKE 'A\%' ESCAPE '\' 查询 A%
  - 空值: IS NULL, IS NOT NULL 不能用 = NULL 或 <> NULL

- 逻辑运算符: `AND`, `OR`, `NOT`
  - 优先级: `NOT` > `AND` > `OR`
  - 可使用括号更改优先级
- `GROUP BY` 按照指定列的值分组
- `HAVING` 分组后的条件
- `ORDER BY` 排序
  - 根据多个列排序: 前面的列优先级高

```
SELECT [ALL | DISTINCT] <column_name_expr> [ [AS]
<column_alias_name>], ...
FROM <table_name|view_name>, ... | (SELECT ...) [AS <alias_name>]
WHERE <condition>
GROUP BY <column_name>, ...
HAVING <condition>
ORDER BY <column_name> [ASC | DESC]
```

## 聚集函数

- `COUNT(*)` : 统计行数
- `COUNT([DISTINCT|ALL] <column_name>)` : 统计列数, 默认为 `ALL`
- `SUM/AVG/MIN/MAX`
- 除了 `COUNT(*)`, 其他聚集函数跳过空值

## 对查询结果分组

- `GROUP BY` : 分组
- 对查询结果分组后, 聚集函数将分别作用于每个组
- `HAVING` 作用于分组后的结果, `WHERE` 作用于分组前的所有结果
  - 例: 查询平均成绩大于等于90分的学生学号和平均成绩

```
SELECT SNO, AVG(SCORE) AS AVG_SCORE
FROM SC
GROUP BY SNO
HAVING AVG(SCORE) >= 90
```

## 连接

```
SELECT Student.*, SC.*
FROM Student, SC
WHERE Student.SNO = SC.SNO
```

-- 自身连接

```
SELECT FIRST.Cno, SECOND.Cpno
FROM C FIRST, C SECOND
WHERE FIRST.Cpno = SECOND.Cno
```

## 集合查询

- `SELECT xxx op SELECT yyy`
- `UNION`：并集，去重
- `UNION ALL`：并集，保留重复值
- `INTERSECT`：交集
- `EXCEPT`：差集
- 参与集合操作的查询结果必须列数相同，数据类型相同

## 插入

```
INSERT INTO <table_name> [<column_name>, ...]
VALUES (<value>, ...)
```

## 更新

```
UPDATE <table_name>
SET <column_name> = <value>, ...
WHERE <condition>
```

## 删除

```
DELETE FROM <table_name>
WHERE <condition>
```

## 空值

- `NULL` 参与算术计算时，结果为 `NULL`
- `NULL` 参与比较时，结果为 `UNKNOWN`
- 含有 `UNKNOWN` 的逻辑运算: `TRUE > UNKNOWN > FALSE`
  - `NOT UNKNOWN` : `UNKNOWN`
  - `AND` : 取“小”的
  - `OR` : 取“大”的

## 建立视图

```
CREATE VIEW <view_name> [<column_name>, ...]
AS <query>
[WITH CHECK OPTION]
```

- `<query>` : 任意查询语句 `SELECT ... FROM ... WHERE ...`
- `WITH CHECK OPTION` : 检查视图的完整性约束，视图的插入、删除、更新操作必须满足视图定义的谓词条件 (`WHERE` 条件)

```
CREATE VIEW F_Student (F_Sno, name, sex, age, dept) AS
SELECT *
FROM Student
WHERE Ssex = '女';
```

上一页  
09-并发

下一页  
编译原理

最后更新于4个月前

