计算机保研知识点整理 数据库篇

计算机考研复试面试常问问题 数据库篇

1.事务

2.并发一致性问题

3.封锁

4.关系数据库设计理论

5.ER 图

1. 事务

概念：事务指的是满足 ACID 特性的一组操作，可以通过 Commit 提交一个事务，也可以使用 Rollback 进行回滚。

ACID 特性：

（1）原子性（Atomicity）：事务被视为不可分割的最小单元，事务的所有操作要么全部提交成功，要么全部失败回滚。回滚可以用回滚日志来实现，回滚日志记录着事务所执行的修改操作，在回滚时反向执行这些修改操作即可。

（2）一致性（Consistency）：数据库在事务执行前后都保持一致性状态。在一致性状态下，所有事务对一个数据的读取结果都是相同的。

（3）隔离性（Isolation）：一个事务所做的修改在最终提交以前，对其它事务是不可见的。

（4）持久性（Durability）：一旦事务提交，则其所做的修改将会永远保存到数据库中。即使系统发生崩溃，事务执行的结果也不能丢失。

使用重做日志来保证持久性。

2. 并发一致性问题

丢失数据

丢失数据： T1 和 T2 两个事务都对一个数据进行修改，先修改 T1 随后修改 T2， T2 的修改覆盖了 T1 的修改。简记为同时修改 。

读脏数据

读脏数据：T1 对一个数据做了修改，T2 读取这一个数据。若 T1 执行 ROLLBACK 操作，则 T2 读取的结果和第一次的结果不一样。简记为 读取失败的修改 。最简单的场景是修改完成后，紧接着查询检验结果。

不可重复读

不可重复读：T2 读取一个数据，T1 对该数据做了修改。如果 T2 再次读取这个数据，此时读取的结果和第一次读取的结果不同。简记为 读时修改 ，重复读取的结果不一样。

幻影读

幻影读：T1 读取某个范围的数据，T2 在这个范围内插入新的数据，T1 再次读取这个范围的数据，此时读取的结果和和第一次读取的结果不同。简记为 读时插入 ，重复读取的结果不一样。

3. 封锁

封锁粒度

MySQL 中提供了两种封锁粒度： 行级锁 以及 表级锁 。

应尽量只锁定需要修改的那部分数据，而不是所有的资源。锁定的数据量越少，发生锁争用的可能就越小，系统的并发程度就越高。但是加锁需要消耗资源，锁的各种操作（包括获取锁、释放锁、以及检查锁状态）都会增加系统开销。因此封锁粒度越小，系统开销就越大。为此，我们在选择封锁粒度时，需在 锁开销 和 并发程度 之间做一个 权衡 。

封锁类型

（1）读写锁

排它锁 (Exclusive)，简写为 X 锁 ，又称 写锁 。

共享锁 (Shared)，简写为 S 锁 ，又称 读锁 。

有以下两个规定：

一个事务对数据对象 A 加了 X 锁，就可以对 A 进行读取和更新。加锁期间其它事务不能对 A 加任何锁。

一个事务对数据对象 A 加了 S 锁，可以对 A 进行读取操作，但是不能进行更新操作。加锁期间其它事务能对 A 加 S 锁，但是不能加 X 锁。

（2）意向锁

使用意向锁 (Intention Locks)，可以更容易地支持多粒度封锁，使得行锁和表锁能够共存。

在存在行级锁和表级锁的情况下，事务 T 想要对表 A 加 X 锁，就需要先检测是否有其它事务对表 A 或者表 A 中的任意一行加了锁，那么就需要对表 A 的每一行都检测一次，这是非常耗时的。

意向锁在原来的 X/S 锁之上引入了 IX / IS，IX / IS 都是 表级别的锁 ，用来表示一个事务稍后会对表中的某个数据行上加 X 锁或 S 锁。整理可得以下两个规定：

一个事务在获得某个数据行对象的 S 锁之前，必须先获得表的 IS 锁或者更强的锁；

一个事务在获得某个数据行对象的 X 锁之前，必须先获得表的 IX 锁

封锁协议

三级封锁协议

一级封锁协议 ：事务 T 要修改数据 A 时必须加 X 锁，直到 T 结束才释放锁。 防止同时修改 ，可解决丢失修改 问题，因不能同时有两个事务对同一个数据进行修改，那么事务的修改就不会被覆盖。

二级封锁协议 ：在一级的基础上，要求读取数据 A 时必须加 S 锁，读取完马上释放 S 锁。防止修 改时读取 ，可解决 丢失修改 和 读脏数据 问题，因为一个事务在对数据 A 进行修改，根据 1 级封锁协议，会加 X 锁，那么就不能再加 S 锁了，也就是不会读入数据。

三级封锁协议 ：在二级的基础上，要求读取数据 A 时必须加 S 锁，直到事务结束了才能释放 S 锁。 防止读取时修改 ，可解决 丢失修改 和 读脏数据 问题，还进一步防止了 不可重复读 的问题，因为读 A 时，其它事务不能对 A 加 X 锁，从而避免了在读的期间数据发生改变。

4. 关系数据库设计理论

函数依赖

- 记 $A \to B$ 表示 A 函数决定 B，也可以说 B 函数依赖于 A。
- 若 $\{A_1, A_2, \ldots, A_n\}$ 是关系的一个或多个属性的集合，该集合函数决定了关系的其它所>有属性并且是 最小的，那么该集合就称为 键码。
- 对于 $A \to B$，如果能找到 A 的真子集 A'，使得 $A' \to B$，那么 $A \to B$ 就是 部分函数>依赖，否则就是 完全函数依赖。
- 对于 $A \to B$，$B \to C$，则 $A \to C$ 是一个 传递函数依赖。

范式

范式理论是为了解决以上提到四种异常。高级别范式的依赖于低级别的范式，1NF 是最低级别的范式。

第一范式（1NF）

属性不可分。即数据库表的每一列都是不可分割的基本数据项，同一列中不能有多个值，即实体中的某个属性不能有多个值或者不能有重复的属性。

第二范式（2NF）

每个非主属性完全函数依赖于键码。可以通过分解来满足 2NF。

分解前:

Sno	Sname	Sdept	Mname	Cname	Grade
1	学生-1	学院-1	院长-1	课程-1	90
2	学生-2	学院-2	院长-2	课程-2	80
2	学生-2	学院-2	院长-2	课程-1	100
3	学生-3	学院-2	院长-2	课程-2	95

以下学生课程关系中，{Sno，Cname} 为键码，有如下函数依赖：

Sno －> Sname，Sdept

Sdept －> Mname

Sno，Cname－> Grade

函数依赖状况分析：

Grade 完全函数依赖于键码，它没有任何冗余数据，每个学生的每门课都有特定的成绩。

Sname，Sdept 和 Mname 都部分依赖于键码，当一个学生选修了多门课时，这些数据就会出现多次，造成大量冗余数据。

分解后

关系 1

Sno	Sname	Sdept	Mname
1	学生-1	学院-1	院长-1
2	学生-2	学院-2	院长-2
3	学生-3	学院-2	院长-2

有以下函数依赖：

Sno –> Sname, Sdept

Sdept –> Mname

关系 2

Sno	Cname	Grade
1	课程-1	90
2	课程-2	80
2	课程-1	100
3	课程-2	95

有以下函数依赖： Sno，Cname –>Grade

第三范式（3NF）
非主属性不传递函数依赖于键码。简而言之，第三范式就是属性不依赖于其它非主属性。
上面的 关系 1 中存在以下传递函数依赖：Sno–> Sdept–> Mname。
分解后

- 关系-1.1：

Sno	Sname	Sdept
1	学生-1	学院-1
2	学生-2	学院-2
3	学生-3	学院-2

- 关系-1.2：

Sdept	Mname
学院-1	院长-1
学院-2	院长-2

5. ER 图

实体关系图（Entity—Relationship，E—R），有三个组成部分：实体、属性、联系。用来进行关系型数据库系统的概念设计。
实体 ：用矩形表示，矩形框内写明实体名.
属性 ：用椭圆形表示，并用无向边将其与相应的实体连接起来。
联系 ：用菱形表示，菱形框内写明联系名，并用无向边分别与有关实体连接起来，同时在无向边旁标上联系的类型（1...1，1...* 或 *...*）就是指存在的三种关系（一对一、一对多或
多对多）。
ER 模型转换为关系模式的原则：
一对一 ：遇到一对一关系的话，在两个实体任选一个添加另一个实体的主键即可。
一对多 ：遇到一对多关系的话，在多端添加另一端的主键。
多对多 ：遇到多对多关系的话，我们需要将联系转换为实体，然后在该实体上加上另外两个实体的主键，作为联系实体的主键，然后再加上该联系自身带的属性即可。