数据库系统原理

教程：数据库系统概论（第5版）

结合：CMU 15-445/645 INTRO TO DATABASE SYSTEMS

华中科技大学 计算机学院

左琼

第十一章　并发控制

Principles of Database Systems

第11章 并发控制概述

- 三级封锁协议
- 可串行性：冲突可串行性
- 两段锁协议
- 遵守第三级封锁协议必然遵守两段锁协议

11.7 两段锁协议（2PL: two-phase locking）

1. 两段锁协议目标

可串行性是并发调度正确性的唯一准则；

2PL是保证并发操作调度的可串行化而提供的封锁协议。

2. 两段锁含义

事务分为两个阶段对数据加锁和解锁：

第一阶段称为扩展阶段（获得锁）；

第二阶段称为收缩阶段（释放锁）。

3. 策略

① 在对任何数据读、写之前，须先获得该数据的锁。

② 在释放一个封锁之后，该事务不能再申请任何其它锁。

11.7 两段锁协议

【例】

- 事务1: 遵循两段锁协议的事务，其封锁序列为：

SLOCK(A) SLOCK(B) XLOCK(C) UNLOCK(B) UNLOCK(A) UNLOCK(C)

扩展阶段　　收缩阶段

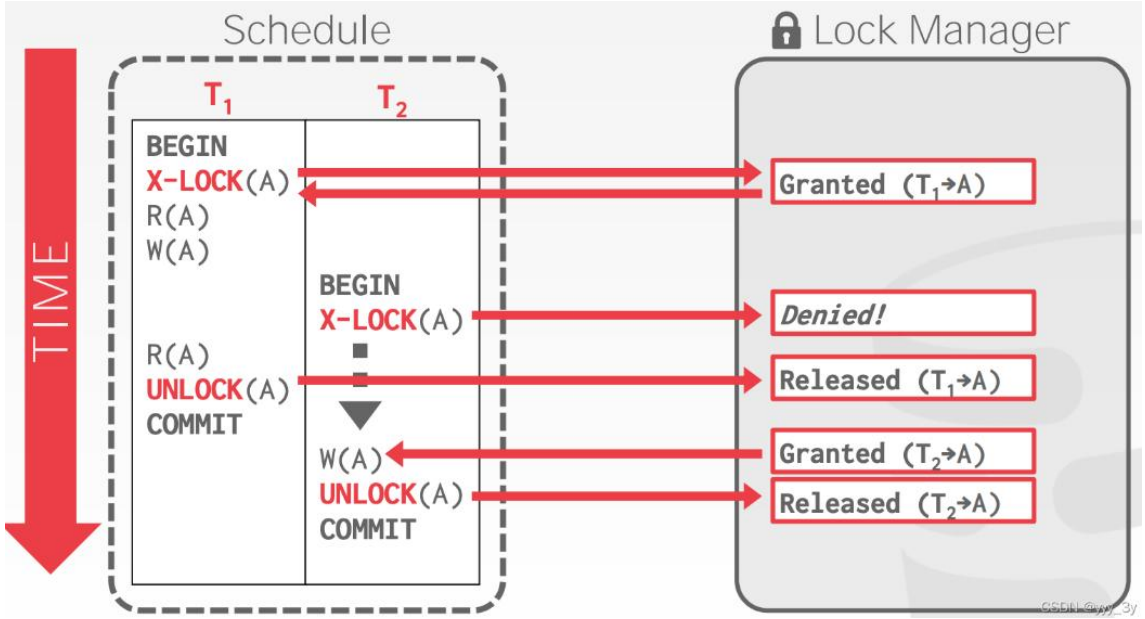
事务	加锁/解锁处理
begin;	
insert into test	加insert对应的锁
update test set...	加update对应的锁
delete from test	加delete对应的锁
commit;	事务提交时，同时释放insert、update、delete对应的锁

- 事务2: 不遵循两段锁协议的事务，其封锁序列为：

SLOCK(A) UNLOCK(A) SLOCK(B) XLOCK(C) UNLOCK(B) UNLOCK(C)

11.7 两段锁协议

- 定理：若所有并发事务都遵守两段锁协议，则对这些事务的所有并发调度都是可串行化的。



- 关于两段锁协议的两点说明：

1）两段锁协议是可串行化调度的充分条件，但不是必要条件。

2）所有遵守2PL的事务，其并行执行的结果一定是正确的。

但，2PL不能防止死锁（注意区别于防止死锁的一次封锁法）。

例

T_1	T_2
SLock B=T B=2 Y=B XLcok A=T	
	SLock A=F 等待
A=Y+1 写回A=3 UNLock B UNLock A	等待
	SLock A=T A=3 Y=A
A＝3 B＝4	XLock B=T B=Y+1 写回B=4 UNLockB UNLockA

遵守2PL，可串行

T_1	T_2
SLock B=T B=2 Y=B Unlock B XLcok A=T	
	SLock A=F 等待
A=Y+1 写回A=3 UNLock A	等待
	SLock A=T A=3 Y=A
A＝3 B＝4	XLock B=T B=Y+1 写回B=4 UNLockB UNLockA

不遵守2PL，可串行

T_1	T_2
	SLock A=T A=2 X=A Unlock A
SLock B=T B=2 Y=B UnLcok B	XLock B=F 等待
	XLock B=T B=X+1 写回B=3 UNLockB
XLock A=T A=Y+1 写回A=3 UNLock A	A＝3 B＝3

不遵守2PL，不可串行

11.7 两段锁协议

- 两段锁协议与防止死锁的一次封锁法:
  - 一次封锁法要求每个事务必须一次将所有要使用的数据全部加锁，否则就不能继续执行，因此一次封锁法遵守两段锁协议。

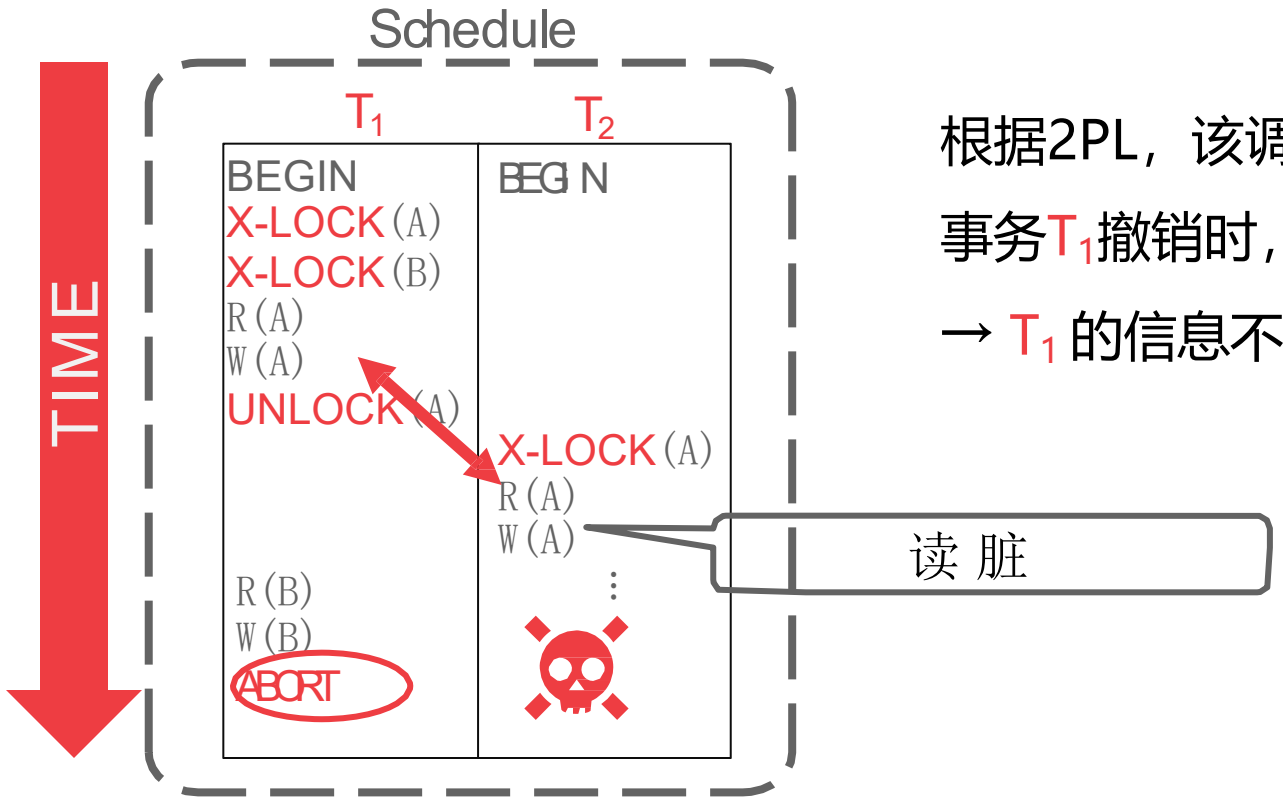
- **但是两段锁协议并不要求事务必须一次将所有要使用的数据全部加锁，因此遵守两段锁协议的事务可能发生死锁**

T_1	T_2
Slock B	
R(B)=2	
	Slock A
	R(A)=2
Xlock A	
等待	Xlock B
等待	等待

遵守两段锁协议的事务可能发生死锁

两段锁协议

- 2PL足够保证冲突可串行化（所产生的调度图是无环），但存在级联撤销（cascading abort）。

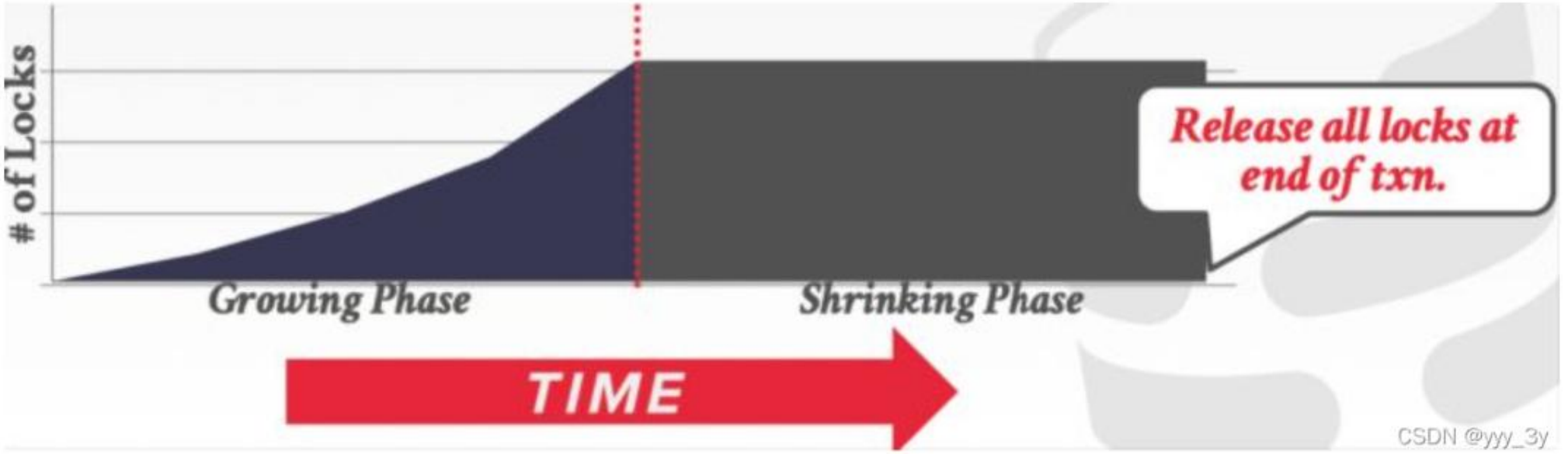


根据2PL，该调度是可行的，但当事务T_1撤销时，数据库须撤销T_2

→ T_1 的信息不能泄露给外界

两段锁协议

- STRONG STRICT 2PL：规定在 Shrinking Phase 阶段，只有到 commit 或 abort 时才会一次性释放全部锁（可以避免脏读的情况）。
  - 调度是严格的：如果在一事务结束前，该事务所写的值不会被其它事务读或写。
  - 优点：不会引发级联撤销，撤销事务只需恢复修改元组的原始值.



两段锁协议 vs. 三级封锁协议

- 两类不同目的的协议:
  - 两段锁协议：
    保证并发调度的正确性。
  - 三级封锁协议：
    在不同程度上保证数据一致性。

- 遵守第三级封锁协议必然遵守两段锁协议。

2PL课堂练习

考虑以下两个事务：

T1：

```
R(B);
R(A);
If A=0 then B=B+ 5 ;
W(B);
```

T2：

```
R(A);
R(B);
If B=0 then A=A+1    else A=A+B;
W(A);
```

(1) 请给T1和T2增加加锁和解锁指令，使他们遵从2PL协议；

(2) 使用上述锁机制后，可能发生死锁么？若有请给出其调度的一个例子。

2PL课堂练习-参考解答

（1）参考如右图。

（2）可能发生死锁。
上述调度T2的SLock(B)提前到T1的XLock(B)之前，就会锁住T1，进而T2的XLock(A)申请不得，锁住T2，成为死锁。

T1	T2
SLock(B)=T;	
R(B);	
SLock(A)=T;	
R(A);	
If A=0 then B=B+5;	SLock(A)=T;
XLock(B)=T;	R(A);
W(B);	SLock(B)=F;
UNLOCK(A);	…
UNLOCK(B);	…
UNLOCK(B);	…
	SLock(B)=T;
	R(B);
	If B=0 then A=A+1 else A=A+B;
	XLock(A)=T;
	W(A);
	UNLOCK(B);
	UNLOCK(A);
	UNLOCK(A);

11.8 封锁的粒度

- 封锁对象的大小称为封锁粒度(Granularity)
- 封锁的对象：逻辑单元，物理单元。

在关系数据库中，封锁对象：

- 逻辑单元: 属性值、属性值集合、元组、关系、索引项、整个索引、整个数据库等
- 物理单元：页（数据页或索引页）、物理记录等。

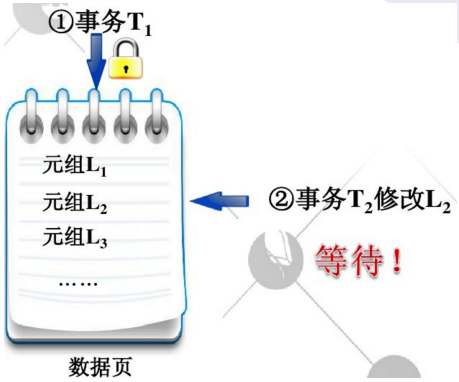
- 封锁粒度与系统的并发度和并发控制的开销密切相关。
  - 封锁的粒度越大，数据库所能够封锁的数据单元就越少，并发度就越小，系统开销也越小；
  - 封锁的粒度越小，并发度较高，但系统开销也就越大

11.8 封锁的粒度

系统开销与封锁粒度有关，也与事务的操作对象相关。

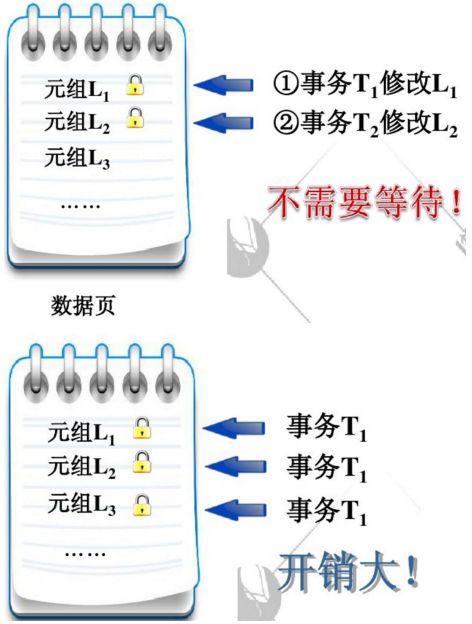
例：

- 若封锁粒度是数据页，事务T1需要修改元组L1，则T1必须对包含L1的整个数据页A加锁。如果T1对A加锁后事务T2要修改A中元组L2，则T2被迫等待，直到T1释放A。



- 如果封锁粒度是元组，则T1和T2可以同时对L1和L2加锁，不需要互相等待，提高了系统的并行度。

  但：事务T需要读取整个表，若封锁粒度是元组，T必须对表中的每一个元组加锁，开销极大



11.8 封锁的粒度

- 多粒度封锁(Multiple Granularity Locking)

  在一个系统中同时支持多种封锁粒度供不同的事务选择。

- 选择封锁粒度的一般原则：

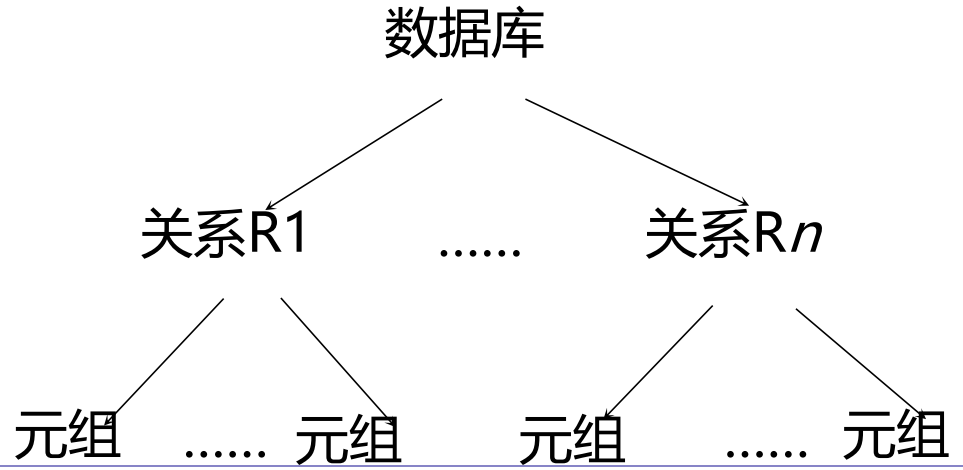
  同时考虑封锁开销和并发度两个因素，适当选择封锁粒度：

  - 需要处理多个关系的大量元组的用户事务：以数据库为封锁单位；
  - 需要处理大量元组的用户事务：以关系为封锁单元；
  - 只处理少量元组的用户事务：以元组为封锁单位。

11.8.1 多粒度封锁

- 多粒度树
  - 以树形结构来表示多级封锁粒度
  - 根结点是整个数据库，表示最大的数据粒度
  - 叶结点表示最小的数据粒度

- 例：三级粒度树。根结点为数据库，数据库的子结点为关系，关系的子结点为元组。



11.8.1 多粒度封锁

多粒度封锁协议：

- 允许多粒度树中的每个结点被独立加锁；
- 对一个结点加锁意味着这个结点的所有后裔结点也被加以同样类型的锁。
- 在多粒度封锁中一个数据对象可能以两种方式封锁：显式封锁和隐式封锁
  - 显式封锁：直接加到数据对象上的封锁。
  - 隐式封锁：该数据对象没有独立加锁，是由于其上级结点加锁而使该数据对象加上了锁；
  - 显式封锁和隐式封锁的效果是一样的。

显式封锁和隐式封锁

- 系统检查封锁冲突时，要检查：
  - 显式封锁
  - 隐式封锁
- 对某个数据对象加锁，系统要检查：
  - 该数据对象：有无显式封锁与之冲突；
  - 所有上级结点：检查本事务的显式封锁是否与该数据对象上的隐式封锁冲突（由上级结点已加的封锁造成的）；
  - 所有下级结点：看上面的显式封锁是否与本事务的隐式封锁（将加到下级结点的封锁）冲突。

11.8.2 意向锁

- 引进意向锁（intention lock）目的：
  提高对某个数据对象加锁时系统的检查效率。
- 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；
- 对任一结点加基本锁，必须先对它的上层结点加意向锁；
  例如，对任一元组加锁时，必须先对它所在的数据库和关系加意向锁。
- 常用意向锁：
  - 意向共享锁(Intent Share Lock，简称IS锁)
  - 意向排它锁(Intent Exclusive Lock，简称IX锁)
  - 共享意向排它锁(Share Intent Exclusive Lock，简称SIX锁)

11.8.2 意向锁

- 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。
- 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。
- 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。
- 加锁规则：
  1）每一个 tx_n 获得锁的顺序都是从上到下，释放锁的顺序是从下到上。
  2）一个节点获得 S 或 IS 时，其所有父节点获得 IS。
  3）一个节点获得 X 或 IX 或 SIX 时，其所有父节点获得 IX。

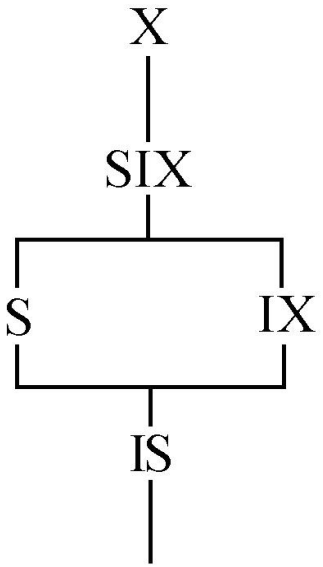
T_1 \ T_2	S	X	IS	IX	SIX	-
S	Y	N	Y	N	N	Y
X	N	N	N	N	N	Y
IS	Y	N	Y	Y	Y	Y
IX	N	N	Y	Y	N	Y
SIX	N	N	Y	N	N	Y
-	Y	Y	Y	Y	Y	Y

Y=Yes，表示相容的请求　　N=No，表示不相容的请求

(a) 数据锁的相容矩阵

11.8.2 意向锁

- 锁的强度
  - 锁的强度是指它对其他锁的排斥程度
  - 一个事务在申请封锁时以强锁代替弱锁是安全的，反之则不然



(b) 锁的强度的偏序关系

11.8.2 意向锁

- 具有意向锁的多粒度封锁方法：
  - 申请封锁时应该按自上而下的次序进行
  - 释放封锁时则应该按自下而上的次序进行
  - 意向锁的作用：实现表级锁和行级锁共存。

- 例：事务T1要对关系R1加S锁
  - 要首先对数据库加IS锁
  - 检查数据库和R1是否已加了不相容的锁(X或IX)
  - 不再需要搜索和检查R1中的元组是否加了不相容的锁(X锁)

SQL Server意向锁例

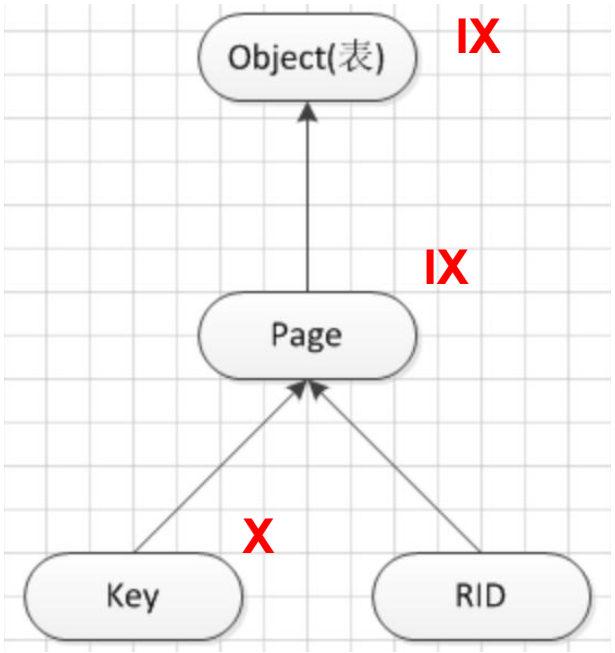
```
create table Student (
    id int,
    name char(30),
    constraint pk_id primary key(id)
) –表上有聚集索引

--插入1000条记录
SET NOCOUNT ON;
GO

DECLARE @i int;
SET @i = 1;
WHILE @i <= 1000 BEGIN
  INSERT INTO Student
values(@i,'zhangsan'+cast(@i as char))
  SET @i = @i + 1;
 END;
GO
```

T1:

```
begin tran
UPDATE Student SET name ='zhangsan' WHERE
id=1000;
```



加行级锁？
加意向锁？
哪个好？

T2: Select * from Student WHERE id >300;

幻象现象

T1: select sum(balance)

from account

where branch_name = ‘Perryridge’

T2: insert into account

values (‘A201’,’Perryridge’, 900)

T1 重复读的过程中，执行T2，插入幻象问题。
怎么办？

- 解决方案：
  - 仅仅封锁元组级别，不够！需要封锁其上的关系
  - 关系上的意向锁
  - Or 索引封锁
  - Or ……

11.8.2 意向锁

- 具有意向锁的多粒度封锁方法：
  - 提高对某个数据对象加锁时系统的检查效率；
  - 提高了系统的并发度；
  - 减少了加锁和解锁的开销；
  - 在实际的数据库管理系统产品中得到广泛应用 。

附*：Mysql上的锁

- 悲观锁（封锁） 和 乐观锁（MVCC，隐藏列，加版本号）
- 表级锁和行级锁：
  - 服务层：只有表级锁；存储引擎层才有行级锁。
- InnoDB的多粒度锁：S，X，IS，IX
  - IS：事务打算给数据行加行共享锁，事务在给一个数据行加共享锁前必须先取得该表的IS锁。
  - IX：事务打算给数据行加行排他锁，事务在给一个数据行加排他锁前必须先取得该表的IX锁。
  - 只有通过索引条件检索数据，InnoDB才使用行级锁，否则，InnoDB将使用表锁。
  - GAP锁：为了防止幻读增加；
  - Next-key锁：行级锁+Gap锁；……

第11章 并发控制总结

- 并行调度可能导致的三类并发错误：
  - 丢失更新、不可重复读、读脏数据
- 并发控制的实质是保证事务的隔离性、一致性。
- 封锁——防止并发错误的并发控制机制。
- 三级封锁协议——从不同程度上防止了并发错误，使系统实现不同程度的事务隔离级别。
- 封锁带来的问题：死锁 vs 活锁
- 可串行化调度——并发调度的正确性标准。判断方法：冲突可串行化。实现方法：两段锁协议。
- 多粒度封锁、意向锁

本章作业：P352 10，11，14

第11章 并发控制总结

串行调度 ⊆ 正确的调度
遵守2PL的调度 ⊆ 可串行化调度 = 正确的调度

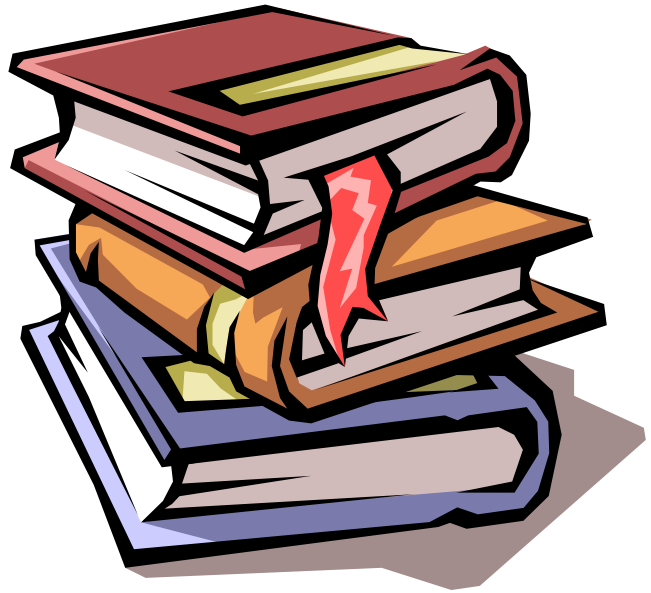
- 并发调度的可串行性：
  - 可串行化调度——并发调度的正确性标准。

    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 。
  - 判断方法：冲突可串行化。

    冲突可串行化——判断可串行化调度的充分条件。还有不满足冲突可串行化条件的可串行化调度。
  - 实现方法：两段锁协议。
- 两段锁协议2PL： 是保证并发操作调度的可串行化而提供的封锁协议。
  - 2PL与可串行性的关系：2PL是可串行化调度的充分条件，但不是必要条件。2PL保证并发调度的正确性。
  - 遵守第三级封锁协议必然遵守两段锁协议。
  - 2PL与死锁的关系：2PL不能防止死锁。

数据库系统概论

总复习 2025



考试说明

- 考试时间：2025.6.23
- 考试形式：闭卷考试
- 平时成绩占30% + 期末考试70%
- 题型：工程认证方式：主观题

DBS章节复习

基础篇

- DB,DBS,DBMS
- 数据模型
- 关系数据库模型
- 关系代数
- SQL语言*
- DB安全性
- DB完整性

设计篇

- 关系数据理论
  - 规范化*
  - Armstrong公理*
  - 模式的分解
- 数据库设计
  - 需求分析
  - 逻辑结构设计*
  - 物理结构设计

系统篇

- 数据库存储*
- 关系处理和查询优化*
  - 代数优化
  - 物理优化
- DB恢复技术*
  - 故障种类
  - 恢复策略
- 并发控制*
  - 封锁协议
  - 并发调度的可串行化

基础篇

- 概念：DB, DBMS, DBS 与 数据库系统组成
- 数据独立性与数据库系统结构（三级模式，两层映像）
- 数据模型
  - 数据模型的三要素：数据结构、数据操作、约束
  - 概念模型：E-R 模型
  - 逻辑模型：层次、网状、关系（区别）
- 关系模型
  - 关系数据结构及定义（关系，候选码，主码，外码，关系模式，关系数据库）。
  - 关系的完整性约束（实体完整性，参照完整性，用户定义的完整性）

基础篇

- 关系代数（5个基本运算（并，差，笛卡儿积，选择，投影）+交，连接，自然连接，除）
- SQL语言四大功能：
  - 数据定义、数据查询、数据更新、数据控制
- 视图的概念及其作用。
- 数据库安全控制：自主存取控制、强制存取控制
- 关系数据库中的三类完整性约束为：
  - 实体完整性约束（主键）
  - 参照完整性约束（外键）
  - 用户定义的完整性约束（NOT NULL，UNIQUE，CHECK，触发器）

设计篇

- 关系数据库理论
  - 函数依赖
  - 关系模式的规范化：1NF，2NF，3NF，BCNF，4NF
  - Armstrong公理系统
    - 逻辑蕴涵，闭包
    - Armstrong公理系统
    - 属性闭包，求key
    - 最小函数依赖集
  - 关系模式分解：无损连接性，依赖保持性
- 数据库设计
  - 需求分析
  - 概念结构设计：E-R图
  - 逻辑结构设计：逻辑模式、外模式
  - 物理设计

系统篇

- 数据库存储：
  - 文件、页、元组存储结构、NSM、DSM
  - B+树、其他索引
  - 缓冲池
- 查询处理：查询处理模型、单表和join的执行方式
- 查询优化：
  - 启发式代数优化：查询优化树
  - 基于规则的存取路径优化：JOIN优化
  - 基于代价估算的优化
- 事务：ACID特性
- 数据库恢复技术
  - 故障：事务故障、系统故障、介质故障
  - 恢复技术：数据库转储、日志、检查点、镜像
- 并发控制
  - 3类并发错误：丢失更新、不可重复读、读脏
  - 封锁、三级封锁协议，死锁
  - 可串行化调度：冲突可串行化、2PL协议
  - 封锁粒度

全面复习，争取好成绩！