第十一章　并发控制

Principles of Database Systems

第11章 并发控制总结

- 数据共享与数据一致性是一对矛盾。
- 数据库的价值在很大程度上取决于它所能提供的数据共享度。
- 数据共享在很大程度上取决于系统允许对数据并发操作的程度。
- 数据并发程度又取决于数据库中的并发控制机制。
- 数据的一致性也取决于并发控制的程度。施加的并发控制愈多，数据的一致性往往愈好。
- 数据库的并发控制以事务为单位。

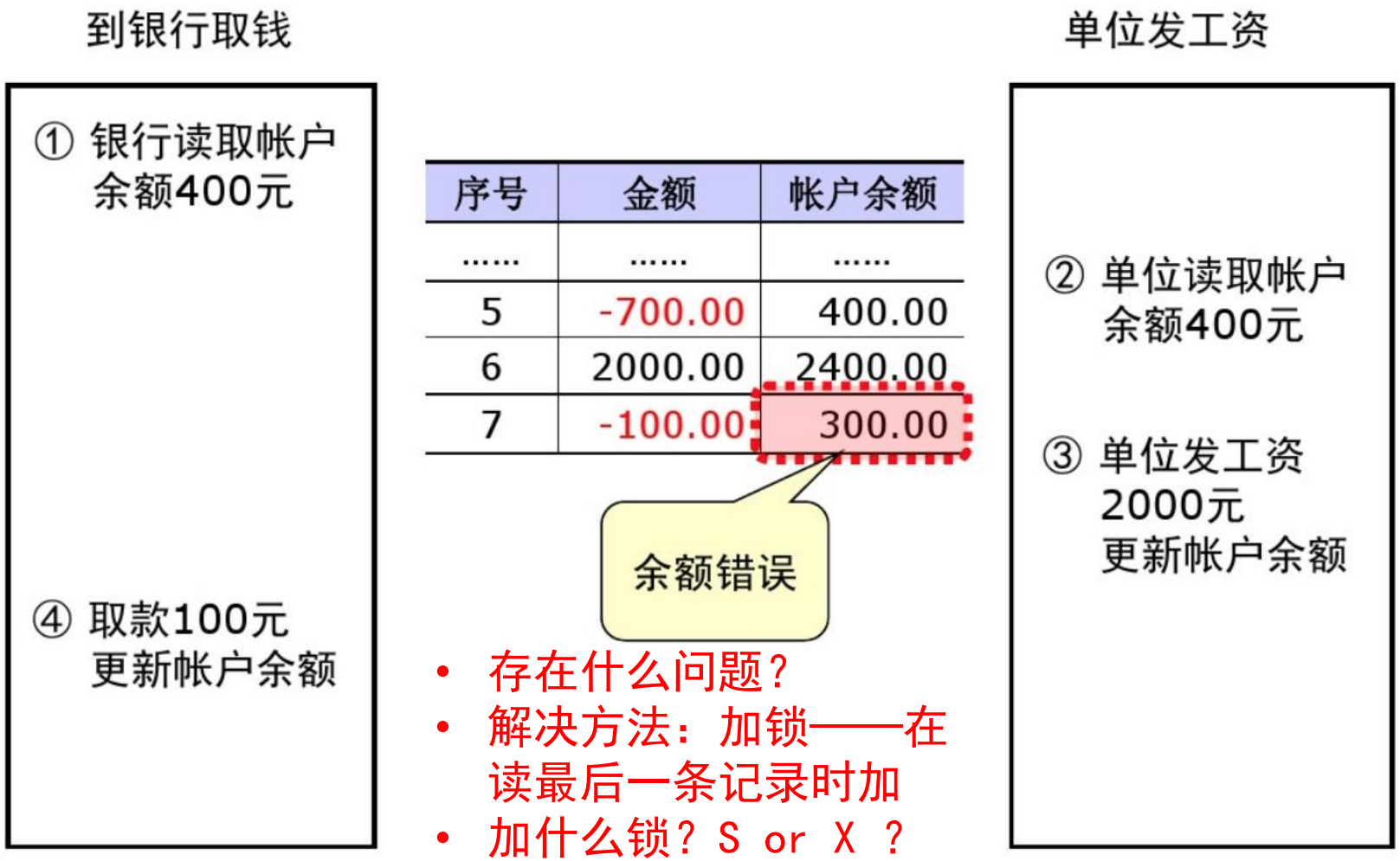
第11章 并发控制总结

	X锁		S锁		一致性保证		
	操作结束释放	事务结束释放	操作结束释放	事务结束释放	不丢失修改	不读脏数据	可重复读
一级		√			√		
二级		√	√		√	√	
三级		√		√	√	√	√

- 并发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。
- 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，不同程度上保证数据一致性。
- 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。
- 两段锁协议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。

串行调度 ⊆ 遵守2PL的调度 ⊆可串行化调度 ⊆ 正确的调度

并发控制例—银行账户



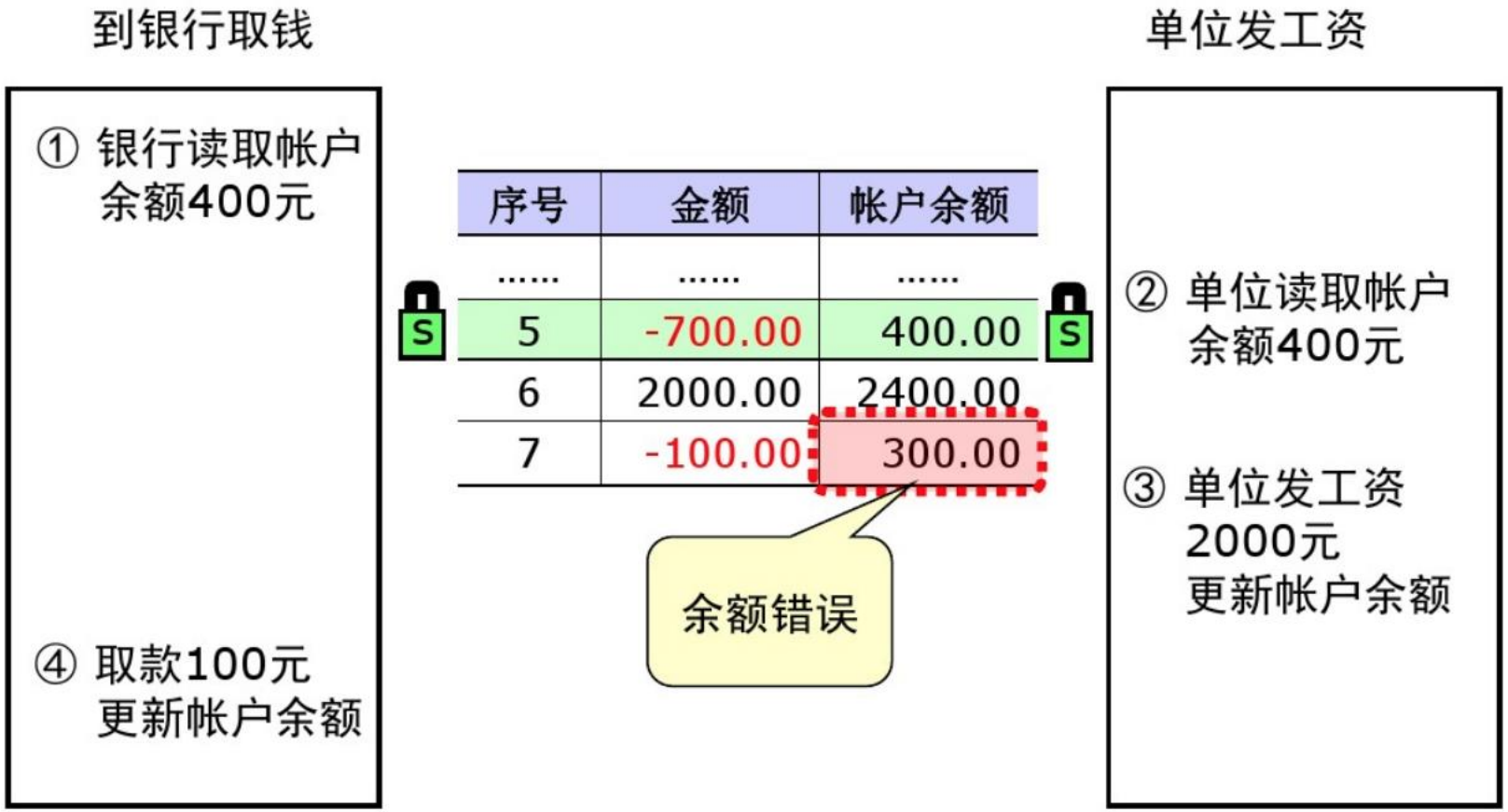
序号	金额	帐户余额
……	……	……
5	-700.00	400.00
6	2000.00	2400.00
7	-100.00	300.00

- 存在什么问题？
- 解决方法：加锁——在读最后一条记录时加
- 加什么锁？S or X ？

并发控制例—银行账户（续）

- 读前上共享锁?

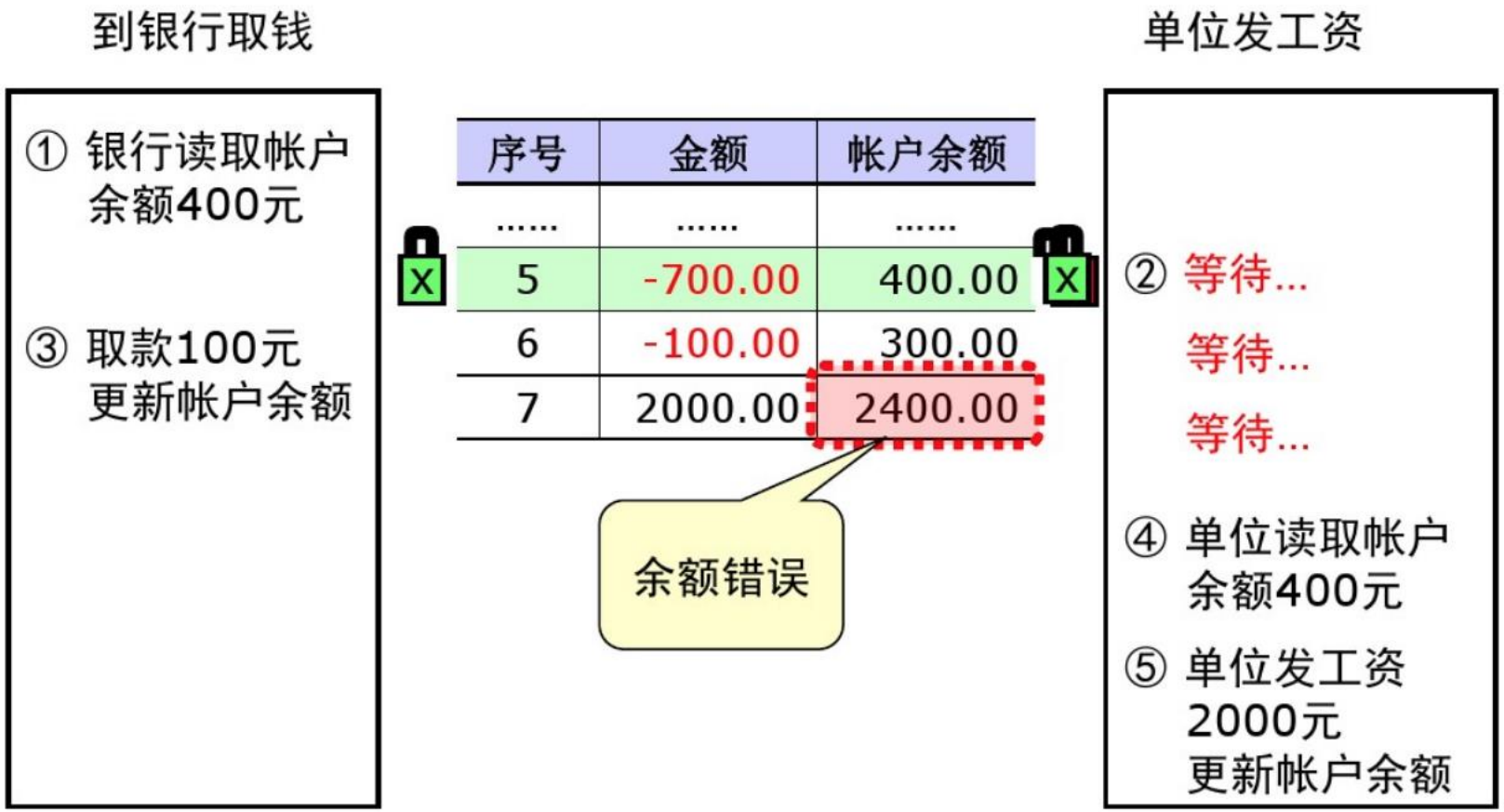
不行！



并发控制例—银行账户（续）

- 读前上排它锁，读完立即释放?

不行！



三级封锁协议

+

客户端程序中的事务定义 **Commit / Rollback**

活锁

1. 含义：事务因故永远处于等待状态。

T_1	T_2	T_3	T_4	T_n
LOCK(R)=T				
	LOCK(R)=F			
	等待	LOCK(R)=F		
UNLOCK(R)	等待			
	等待	LOCK(R)=T		
	等待	…	LOCK(R)=F	
	等待	UNLOCK(R)		
	等待		LOCK(R)=T	

2. 预防方法

FCFS（First Come First Server）：先来先服务

对于事务有优先级的系统，可设置一个最长等待时间，与优先级结合，调度事务的执行。

死锁

1. 含义：两个或两个以上事务均处于等待状态，每个事务都在等待其中另一个事务封锁的数据，导致任何事务都不能继续执行的现象称为死锁。

时间	T_A	T_B
t1	X Lock A	
t2		X Lock B
t3	X Lock B 等待	
t4		X Lock A 等待
t5	等待	等待

死锁实例1

死锁情况：两个事务的两条SQL产生

```
Table T1(id primary key, name)
```

```
Session 1:
Begin;

Select * from t1 where id=1 for update;

Update t1 set name= 'deadlock' where id=5;
```

```
Session 2:
Begin;

Delete from t1 where id=5;

Delete from t1 where id=1;
```

死锁发生

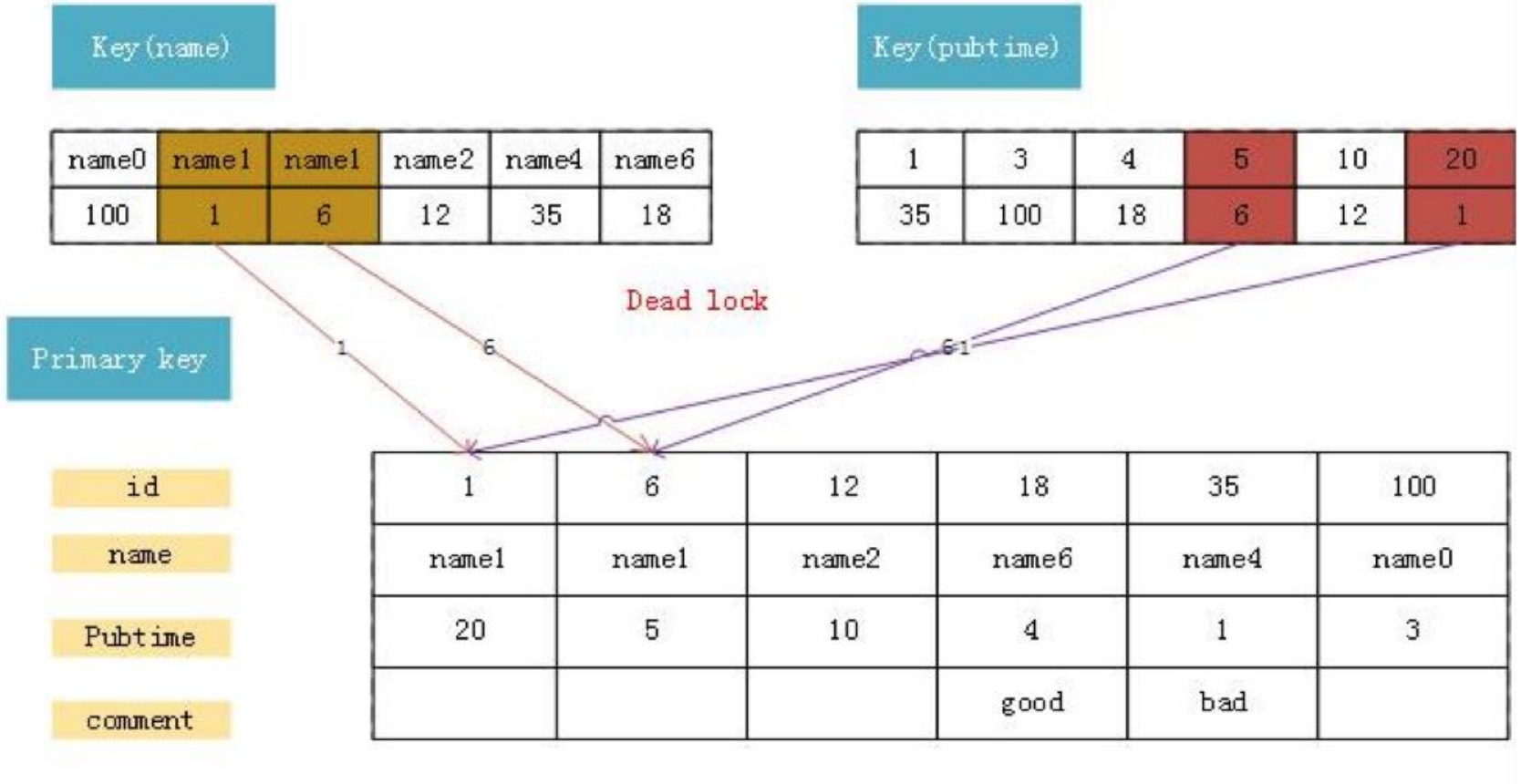
1	2	3	4	5	6
Aaa	bbb	ccc	ddd	eee	ff

死锁实例2

死锁情况：两个事务的一条SQL产生

```
Table T2(id primary key, name key, pubtime key, comment)
```

Session 1:	Session 2:
Begin:	Begin:
Update t2 set comment= 'avd' where name=' name1'	Select * from t2 where pubtime>=5 for update



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。

- 2PL不能防止死锁。

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

可串行化调度习题

设有一个并发事务调度序列如下：

r1(A); w2(A); r2(B); ???; w1(C); w2(B) ……

其中??? 表示某事务的一个读或者写操作（例如r1(B)），请给出 ??? 的所有可能实例，使得该调度序列是一个非冲突可串行化调度。

答案：

r1(A) w1(A) w1(B) r2(C) w2(C)

方法1：列表排除

	A	B	C
R1	非	冲串	冲串
R2	冲串	冲串	非
W1	非	非	冲串
W2	冲串	冲串	非

方法2：
找???前后的冲突操作。
如：
R2(B)：W1(B)

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。提高了系统的并发度，减少了加锁和解锁的开销。
  - 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)

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

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

SQL Server意向锁例

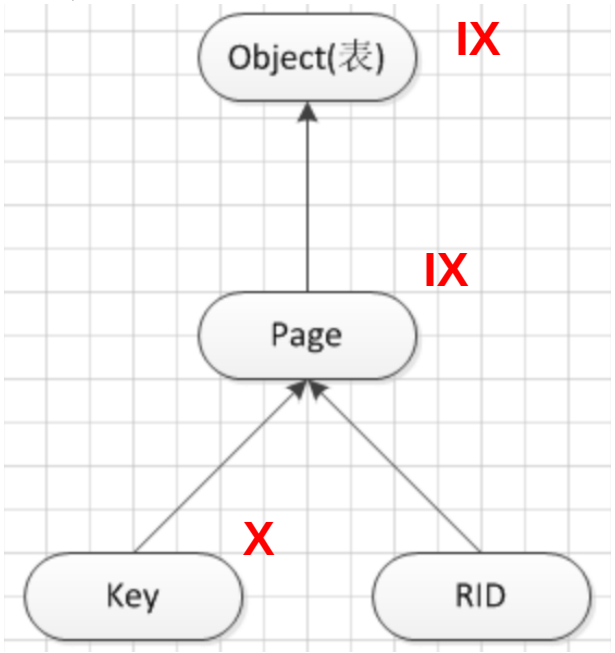

```
create table Student (
    id int,
    name char(30),
    constraint pk_id primary key(id)
) –表上有聚集索引

--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;
```



T2: Select * from Student WHERE id >300;

幻象现象习题

T1: select sum(balance)
from account
where branch_name = ‘Perryridge’

T2: insert into account
values (‘A201’,’Perryridge’, 900)

如何防幻象?

- 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?

附：MySQL事务处理机制

允许脏读，可能读取到其他会话中未提交事务修改的数据。不加锁

只能读取到已经提交的数据。读不加锁，增删改加锁

隔离级别	脏读（Dirty Read）	不可重复读（N eatable Read）	幻读（Phantom Read）
未提交读（Read uncommitted）	可能	可能	可能
已提交读（Read committed）	不可能	可能	可能
可重复读（Repeatable read）	不可能	不可能	可能
可串行化（Serializable ）	不可能	不可能	不可能

完全串行化的读，每次读都需要获得表级共享锁，读写相互都会阻塞。

在同一个事务内的查询都是事务开始时刻一致的，InnoDB默认级别。

附：MySQL事务处理机制

- MySQL RC（已提交读）

事务A	事务B
begin;	begin;
update class_teacher set class_name='初三二班' where teacher_id=1;	update class_teacher set class_name='初三三班' where teacher_id=1;
	ERROR 1205 (HY000): Lock wait timeout exceeded; try restarting transaction
commit;	

teacher_id上如果有索引，加行锁。
否则，给整张表的所有行加行锁。
改进：
过滤条件，发现不满足后，会调用unlock_row方法，把不满足条件的记录释放锁

附：MySQL事务处理机制

- MySQL RC（已提交读）

事务A	事务B
begin;	begin;
select id,class_name,teacher_id from class_teacher where teacher_id=1;	
	update class_teacher set class_name='初三三班' where id=1;
	commit;
select id,class_name,teacher_id from class_teacher where teacher_id=1;	

id	class_name	teacher_id
1	初三二班	1
2	初三一班	1

读不加锁，增删改加锁

id	class_name	teacher_id
1	初三三班	1
2	初三一班	1

不可重复读

计算

附：MySQL事务处理机制

- MySQL RR（可重复读，默认隔离级别）

事务A	事务B	事务C
begin;	begin;	begin;
select id,class_name,teacher_id from class_teacher where teacher_id=1;		

id	class_name	teacher_id
1	初三二班	1
2	初三一班	1

	update class_teacher set class_name='初三三班' where id=1;	
	commit;	
		insert into class_teacher values (null,'初三三班',1);
		commit;
select id,class_name,teacher_id from class_teacher where teacher_id=1;		

id	class_name	teacher_id
1	初三二班	1
2	初三一班	1

幻读，不能通过行锁来避免。
解决方法：
1）可串行化
2）MVCC

可重复读
RR保证对读到的记录加锁，同时，保证对读取的范围加锁，新的满足条件的记录不得插入

MVCC

- **Multi-Version Concurrent Control**

MVCC只在RC和RR两个隔离级别下工作。
非阻塞读；写上行锁。

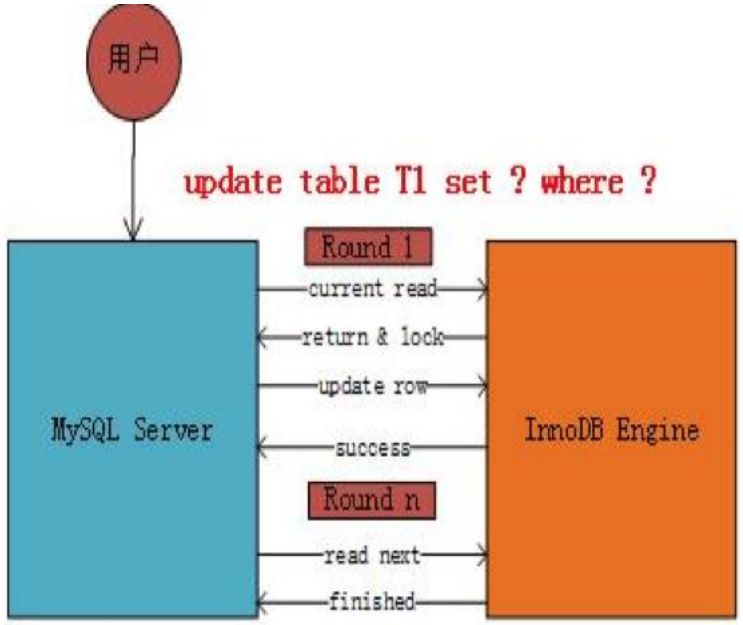
- 读分为2类：
  - **快照读（Snapshot read）**：读取记录的可见版本，不会对返回的记录加锁。

    e.g. select * from t1 where ….

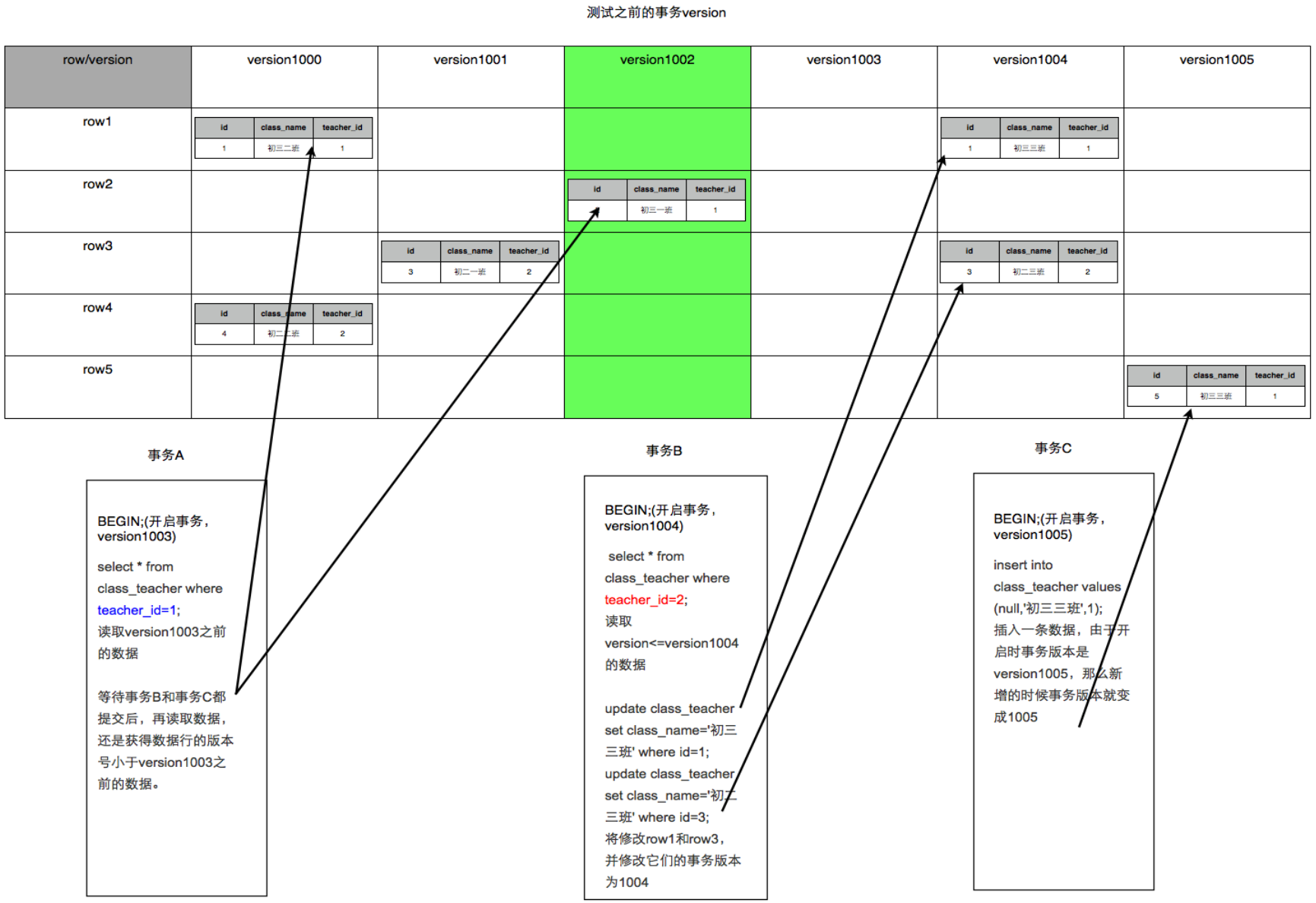
  - **当前读（Current read）**：读取的是记录的最新版本，对返回记录加锁。

    e.g. select * from t2 where ? lock in share mode; / ? for update

    insert, update, delete



MySQL InnoDB MVCC

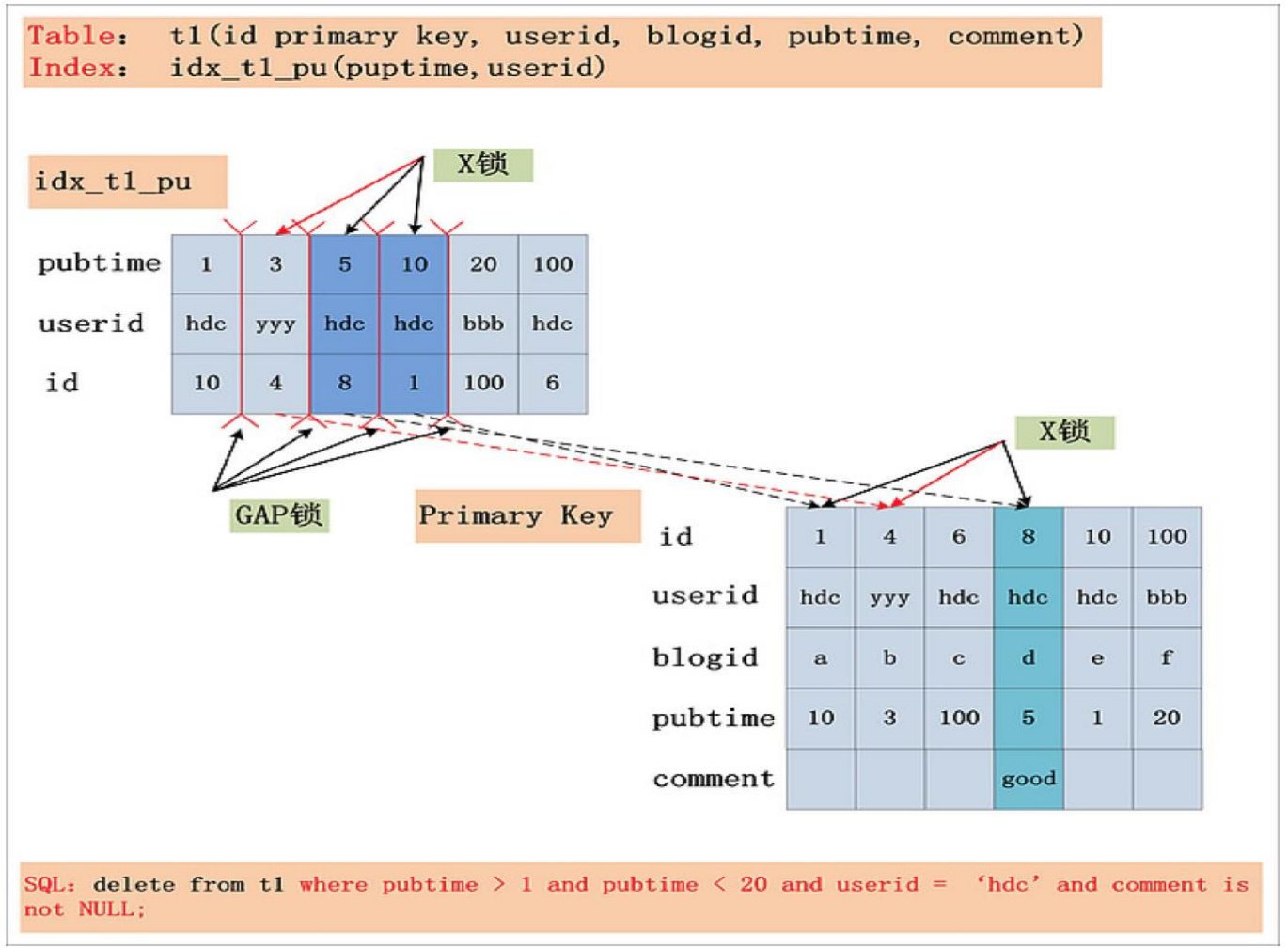


在InnoDB中，会在每行数据后添加两个额外的隐藏的值来实现MVCC，这两个值一个记录这行数据何时被创建，另外一个记录这行数据何时过期（或者被删除）。

MVCC防幻象

- 行锁 + Gap锁
- Gap锁：不允许在数据记录前插入数据。

针对复杂SQL，先提取where条件，index key确定范围。GAP不满足Index Filter的记录，不加X锁，否则加X锁；Table Filter条件，无论是否满足，都要加X锁。



综合练习题

- 假设一个B/S的电商系统中，在做促销活动，每件特价商品限卖100份，买家登陆该网站，浏览特价商品信息（商品表select操作），然后下订单（在select得到的商品表中，勾选商品，填写订单，insert订单表）；若下单成功，则系统后台会修改商品数量（update商品表）。若不考虑触发器、断言技术，请结合你所知道的数据库运行、保护机制回答下列问题。
- （1）简述上述程序在设计上需要应用数据库的什么概念，从而可以利用DBMS对应用程序任务的安全性和完整性之外的保护机制？请说出所涉及到的保护机制，并用专业术语描述需要避免的错误及其类型。

答：

（1）需要定义事务，下订单过程中的查询、插入、更新应属于一个事务。

涉及到事务的恢复机制和并发控制机制。

恢复机制避免出现事务的原子性被破坏、持久性被破坏。

并发控制避免出现丢失更新（下订单人数少于限购记录数）和读脏数据（读取到的订单其他事务之后回滚了）。

综合练习题(续)

- 假设一个B/S的电商系统中，在做促销活动，每件特价商品限卖100份，买家登陆该网站，浏览特价商品信息（商品表select操作），然后下订单（在select得到的商品表中，勾选商品，填写订单，insert订单表）；若下单成功，则系统后台会修改商品数量（update商品表）。若不考虑触发器、断言技术，请结合你所知道的数据库运行、保护机制回答下列问题。
- （2）如果多个人同时购买同一件特价商品，请设计相应的DBMS操作控制协议来避免出现冲突错误，并且尽可能的不降低系统的并发性能。

答：

（2）第2级封锁协议，写操作之前申请排它锁，直到事务提交才释放排它锁；读操作之前申请共享锁，读完之后可马上释放。

数据库系统概论

总复习 2020

考试说明

- 考试时间：复学
- 考试形式：开卷考试
- MOOC成绩20% + 平时成绩占10%？+ 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

- 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
- 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
- 索引 与 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
    - 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

- 查询处理和查询优化
  - 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
  - 可串行化调度：冲突可串行化、两段锁协议
  - 封锁粒度

全面复习，争取好成绩！