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

练习：P327 13

- 考虑如下调度，说明这些调度集合之间的包含关系：
  (1) 正确的调度
  (2) 可串行化调度
  (3) 遵守2PL的调度
  (4) 串行调度

答： (4) $\subseteq$ (3) $\subseteq$ (2) $\subseteq$ (1)

11.6 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

11.6 封锁的粒度

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。

11.6 封锁的粒度

- 例：

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

系统开销与封锁粒度有关，也与事务的操作对象相关。

11.6 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：

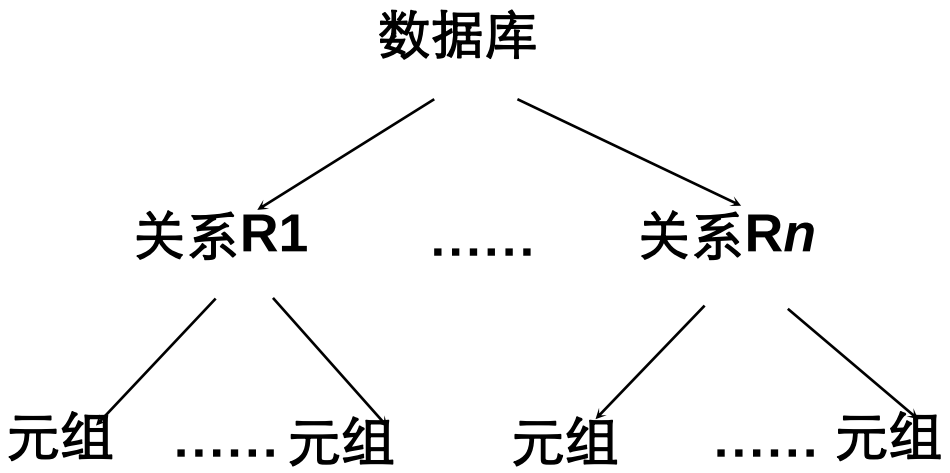
  同时考虑封锁开销和并发度两个因素，适当选择封锁粒度

  - 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

11.6.1 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。



11.6.1 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地加锁；
- 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

11.6.2 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)

11.6.2 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。
  - 例：事务T1要对*R*1中某个元组加S锁，则要首先对关系*R*1和数据库加IS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。
  - 例：事务T1要对*R*1中某个元组加X锁，则要首先对关系*R*1和数据库加IX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。
  - 例：对某个表加SIX锁，则表示该事务要读整个表（所以要对该表加S锁），同时会更新个别元组（所以要对该表加IX锁）。

11.6.2 意向锁

意向锁的相容矩阵

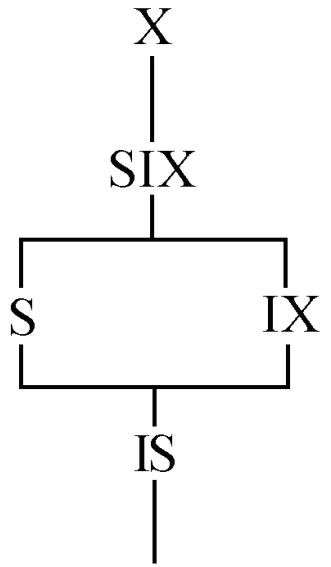
T_1 \ T_2	S	X	IS	IX	SIX	–
S	Y	N	Y	N	N	Y
X	N	N	N	N	N	Y
IS	Y	N	Y	Y	Y	Y
IX	N	N	Y	Y	N	Y
SIX	N	N	Y	N	N	Y
–	Y	Y	Y	Y	Y	Y

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

11.6.2 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



(b) 锁的强度的偏序关系

11.6.2 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

- 例：事务T1要对关系*R*1加S锁
  - 要首先对数据库加IS锁
  - 检查数据库和*R*1是否已加了不相容的锁(X或IX)
  - 不再需要搜索和检查*R*1中的元组是否加了不相容的锁(X锁)

SQL Server意向锁例

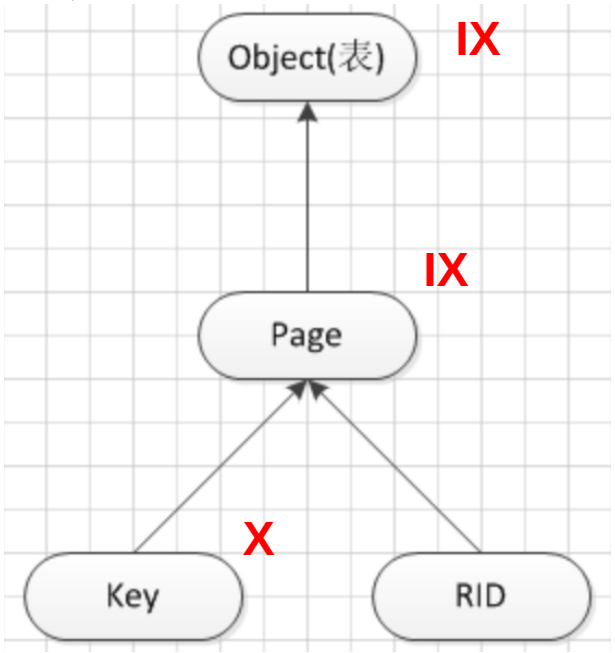

```
create table Student (
    id int,
    name char(30),
    constraint pk_id primary key(id)
) –表上有聚集索引

--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;
```



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)

- 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 ……

11.6.2 意向锁

- 具有意向锁的多粒度封锁方法:
  - 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

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。

本章作业：P326 9，10，14

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。
- 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。
- 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。
- 两段锁协议是可串行化调度的充分条件，但不是必要条件

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[illegible]读（NonRepeatable 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

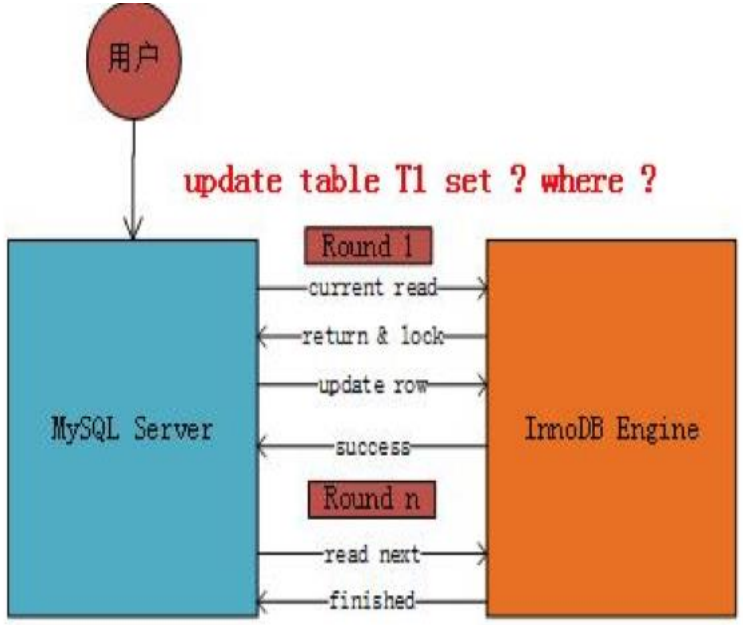
- Multi-Version Concurrent Control
- 读分为2类：
  - 快照读（Snapshot read）：*读取记录的可见版本，不会对返回的记录加锁。*

    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。*

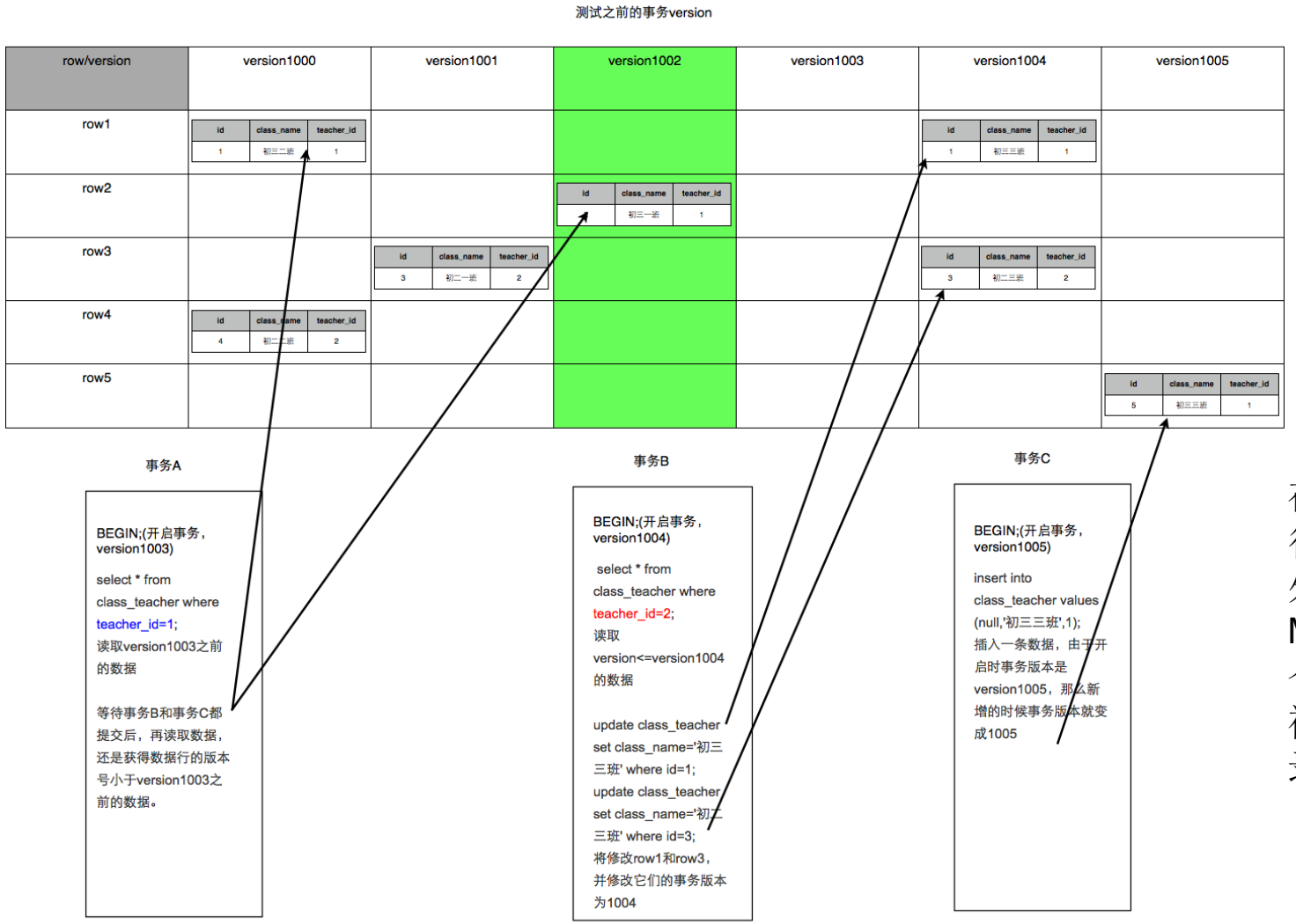
    e.g. select * from t2 where ? lock in share mode; / ? for update

    insert, update, delete

MVCC只在RC和RR两个隔离级别下工作。
非阻塞读；写上行锁。



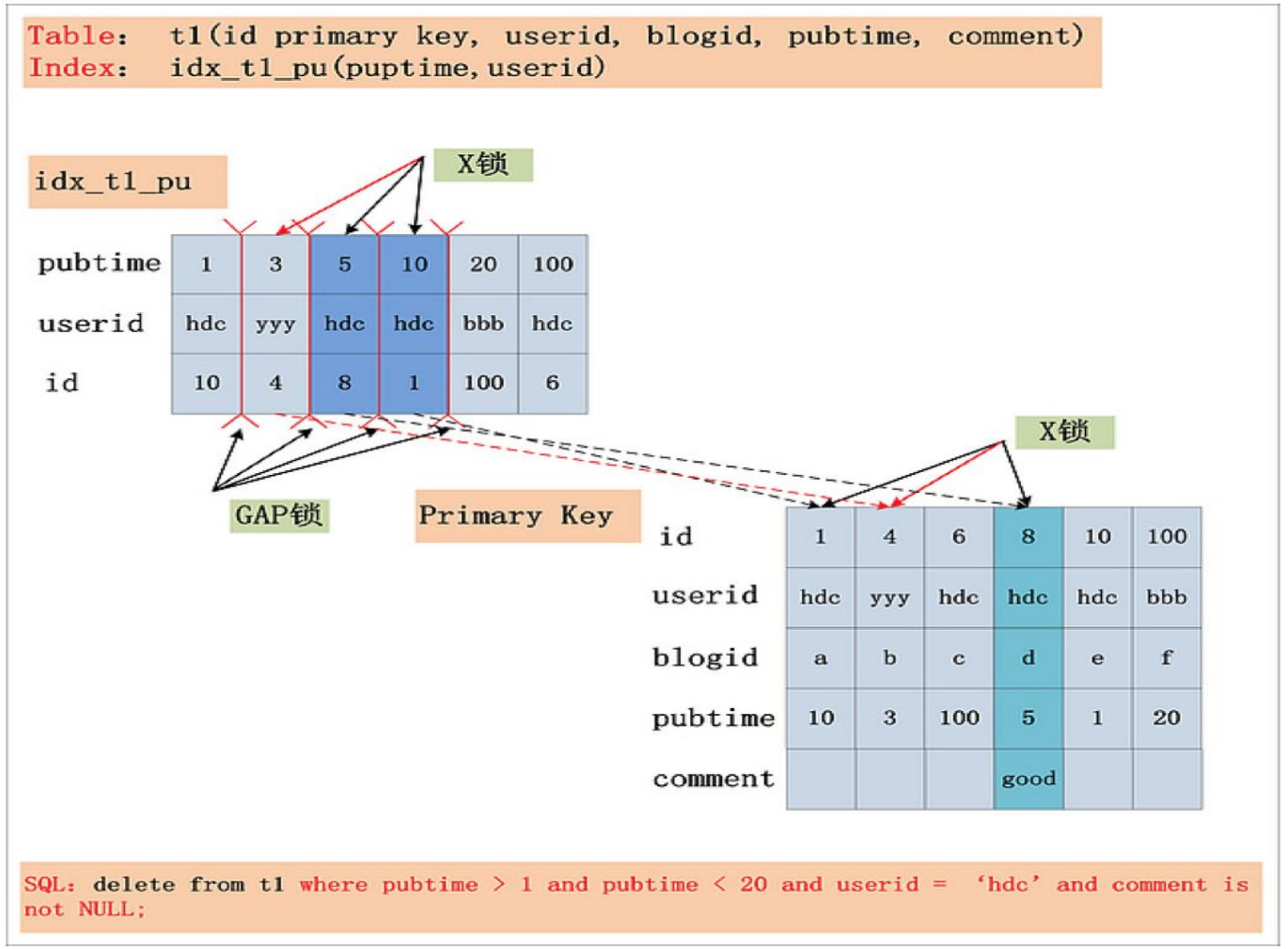
MySQL InnoDB MVCC



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。



课堂练习P326 9

设如下3 个事务:

Tl : A : = A + 2 ;

T2: A : = A * 2 ;

T3: A : = A **2 ; (A <-A*A)

设 A 的初值为 0 。

(1) 若这 3 个事务允许并行执行，则有多少可能的正确结果，请一一列举出来。

(2) 请给出一个可串行化的调度，并给出执行结果

(3) 请给出一个非串行化的调度，并给出执行结果。

(4) 若这 3 个事务都遵守两段锁协议，请给出一个不产生死锁的可串行化调度。

(5) 若这 3 个事务都遵守两段锁协议，请给出一个产生死锁的调度。