高级数据库系统全部作业正确版

版权说明

阅读提示

本 PDF 文件含书签目录，便于读者快速定位。一题多解的，后面的答案（一般是图片格式）可信度准确度更高。

HW-1 (Relational Databases Review & Data Storage)

1 为什么关系数据模型要求一个关系中的元组不能重复？这一要求在 SQL 中是通过什么实现的?

因为关系数据模型中关系是元组的集合，而集合中不能有重复的元素。通过实体完整性即主键实现。

关系模型基于集合论，关系代数以集合运算为基本运算；SQL中采用primary key完整性约束保证元组实体的唯一，也可通过unique key来保证某个键值的唯一性

2 给定学生表 Student(s#,sname,age)、课程表 Course(c#,cname,type,credit)和选课表 SC(s#,c#,score)。其中 type 是整型，0 表示必修课，1 表示选修课。credit 表示课程学分。请用一条 SQL 语句，回答下面的查询

(1) 求出已选必修课总学分大于 16 并且所选必修课成绩都大于 75 分的学生姓名

```
select sname
from Student join SC on Student.s#=SC.s# join Course on
SC.c#=Course.c#
where Course.type=1
group by Student.s#
having sum(credit)>16 and min(score)>75
```

```
select cname, avg(score),
   sum(case when score < 60 then 1 else 0 end) /
   count(*)
from Course C, SC
where C.c# = SC.c#
group by C.c#
```

(2) 求出各门课程的课程名、平均成绩和不及格率（提示：不在选课表中的课程不用考虑）

```
select c.c#, cname, avg(sc.score),
    (select count(*) from SC where SC.c# = c.c# and SC.score
    < 60) / count(sc.score)
from Course c, SC sc
where c.c# = sc.c#
group by c.c#
```

- 求出各门课程的课程名、平均成绩和不及格率（提示：不在选课表中的课程不用考虑）

```
select cname, avgscore, unqualified /total
from Course C,
        (select c# , sum(case when score < 60 then 1 else 0 end) as unqualified from SC group by c# ) as temp1,
        //(select c#, count(*) as unqualified from SC where score < 60 group by c#) as temp1,
        //如果某门课全都及格，这门课会统计不到
        (select c#, count(*) as total , avg(score) as avgscore from SC group by c#) as temp2
where C.c# = SC.c# and temp1.c#=C.c# and temp2.c#=C.c#
```

- Select子句的语法（将返回集作为选择列）不是所有dbms都支持

3 假设某磁盘具有以下特性：

（1）有 8 个盘面和 8192 个柱面

（2）盘面直径为 3.5 英寸，其中内圈不存储数据，内圈直径为 1.5 英寸

（3）每磁道平均有 256 个扇区，每个扇区 512 字节

（4）每个磁道 10%被用于间隙

（5）磁盘转速为 7200 RPM

（6）磁头启动到停止需要 1ms，每移动 500 个柱面另加 1ms

回答下列问题：

（1）磁盘容量是多少?

（2）如果所有的磁道拥有相同的扇区数，那么最内圈的磁道的位密度是多少？

（3）如果一个块是 8KB，那么一个块的传输时间是多少？

（4）平均寻道时间是多少？

（5）平均旋转等待时间是多少？

(1)

$8 \times 8192 \times 256 \times 512Bytes = 2^{33}Bytes = 8GB$

(2)

最内圈直径为 1.5 英寸

$$\frac{256 \times 512\ \ Bytes}{\pi \times 1.5 \times 90\%\ \ in} = 247238.5979\ \ bit/in$$

(3)

$8KB \div 512B/sector = 16\ \ sector$

即经过 16 个扇区和 15 个间隙

7200RPM=7200r/60000ms 即$\frac{25}{3}ms/r$

已知 256 扇区每磁道，10%间隙，则

$$16 \times \frac{1}{256} \times 90\% \times \frac{25}{3}ms + 15 \times \frac{1}{256} \times 10\% \times \frac{25}{3}ms = 0.5176ms$$

(4)

- 线段上任取两点，两点间距离的长度期望是线段长度的 1/3
- 平均寻道数 = 8192 / 3 = 2730
- 平均寻道时间 = 1 + 2730 / 500 = 6.5 ms

(5)

平均旋转等待时间为磁头旋转半圈的时间，即

$$\frac{60000}{7200} \div 2 = 4.17ms$$

- 磁盘转一圈所需时间为：
  - 1 / (7200 RPM) = 8.333 ms
- 平均旋转等待时间为旋转半圈所需的时间
  - 8.333/2 = 4.17 ms

（1） 8192 * 8 * 256 * 512 字节 = 8GB

（2）

- 每个磁道容量

N = 256 * 512 * 8 bits

- 最内圈扇区所占长度

l= 1.5 * π * 90% inch

- 因此最内圈磁道位密度

N/l= 247238.5979 bpi

（3）8KB 为 16 个扇区，因此磁头需要经过 16 个扇区和扇区之间的 15 个间隙，覆盖的总长度占磁道的比值为：

- (16/256) * 90% + (15/256) * 10% = 0.062109
- 磁盘转一圈所需时间为：

1 / (7200 RPM) = 8.333 ms

- 因此一个块的传输时间为：

0.062109 * 8.333 ms = 0.517 ms

4

假设某块磁盘的参数如下：容量为 36.7GB，传输速率为 45MB/s，旋转一圈的时间为 4ms，平均寻道时间为 5ms，最小寻道时间为 0.65ms（指磁头寻道到相邻磁道的时间），一个磁道大小为 180KB。如果磁盘块大小为 4KB，请回答下面问题（所有结果均四舍五入保留小数点后两位）：

（1） 随机读取 1000 个磁盘块需要多少时间（ms）？

（2） 假定（1）中的 1000 个磁盘块在单个磁道上连续存储，并且所有磁盘块存储在相邻的磁道上，此时读取这 1000 个磁盘块需要多少时间（ms）？

(1)

随机读取一个磁盘块需要的时间=平均寻道时间+旋转一圈的时间×$\frac{1}{2}$+传输时间

传输时间=$4ms \times \frac{4}{180} = 0.088889ms$，

此时读取速度为$4\text{KB} \div \frac{0.088889}{1000} s = 43.9MB/s < 45MB/s$，无矛盾

$1000 \times \left(5 + 4 \times \frac{1}{2} + 0.088889\right) = 7088.89ms$

(2)

时间为：

$$\text{平均寻道时间} + \text{旋转一圈的时间} \times \frac{1}{2} + \text{传输时间} \times 1000 + \text{最小寻道时间} \times \left(\left\lceil \frac{1000}{\frac{180}{4}} \right\rceil - 1\right)$$

$$= 5ms + 4 \times \frac{1}{2} + 0.088889 \times 1000 + 0.65 \times 22$$

$$= 110.19\text{ms}$$

（1）

- 由题意知平均寻道时间 t1 = 5 ms，平均旋转时间 t2 = 4 ms / 2 = 2 ms
  传输 1 个磁盘块时间
  - t3 = (4 KB / 180 KB) * 4 ms = 0.09 ms
- 因此随机读取 1000 个磁盘块所需时间为：
  - 1000 * (t1 + t2 + t3) = 7090 ms

（2）

- 连续存储的 1000 个磁盘块占用的磁道数为：⌈1000 * 4 / 180⌉ = 23，因此需要 22 次相邻磁道的最小寻道时间 t4。
  由于磁盘块连续存储，只考虑一次平均寻道时间 t1 和 1 次平均旋转时间 t2，因此读取这 1000 个磁盘块所需时间为：
  - t1 + t2 + 1000 * t3 + 22 * t4 = 111.3 ms

HW-2 (Data Representation)

2. 5. 1

习题 2. 5. 1 假设一条记录有如下顺序的字段：一个长度为 23 的字符串，一个 2 字节整数，一个 SQL 日期，一个 SQL 时间(无小数点)。如果

a)字段可在任何字节处开始，

b)字段必须在 8 的倍数的字节处开始，

c)字段必须在 4 的倍数的字节处开始，

这条记录占用多少个字节？

a) 23+2+10+8=43B

b) 24+8+16+8=56B

c) 24+4+12+8=48B

2. 6. 9

!习题 2. 6. 9 假设我们有 4096 字节块，块中存储 200 字节长的记录。块首部由一个偏移量表组成，如图 2-19 所示，它使用 2 字节长指针指向块内记录。通常，每天向每块插入两条记录，删除一条记录。删除记录必须使用一个“删除标记”代替它的指针，因为可能会有悬挂指针指向它。更明确地说，假设任何一天删除记录总发生在插入之前。如果刚开始时块是空的，多少天之后，不再有插入记录的空间？

每天增加 200×2+2×2=404B，删除 200B。

第 1 天结束 404B，第 2 天结束 404+(404-200)B，第 3 天结束 404+(3-1)×(404-200)B……

第 19 天结束 404+(19-1)×(404-200)B=4076B。

第 20 天删除后占用 3876B，插入 404B 会占用 4280B>4096B(只插入一条还是够的)

所以第 20 天，不再有插入记录的空间。

2. 7. 1

习题 2. 7. 1 一个病人记录包含以下定长字段：病人的出生日期，社会保险号码，病人 ID，每一个字段都是 9 字节长。它还有下列变长字段：姓名，住址和病史。如果记录内一个指针需要 8 字节，记录长度是一个 2 字节整数，不包括变长字段空间，这条记录需要多少字节？你可以假设不需要对字段进行对齐。

3 个定长字段，记录长度，指向后两个变长字段的指针：

3×9+2×8+2=45B

定长字段有3个，每个有9个字节长，所以需要 3×9=27字节。

而记录的首部需要写入记录的长度和指向所有除第一个以外的变长字段起始处的指针。而记录长度2字节，指向“住址”的指针8字节，指向“病史”的指针8字节。所以一共需要27+2+8+8 = 45字节。

期中考试

1

假设磁盘块大小为 8 KB，块中存储 200 字节的定长记录，块首部只包括一个 8 字节的模式指针和一个偏移量表。对于插入块内的每条记录，在偏移量表中都增加一个 2 字节的指针指向该记录。假设每天向块内插入 4 条记录（空间不足时允许插入部分记录后结束全部操作），删除 2 条记录。假设每天的删除记录操作总是发生在插入记录之前，删除记录使用一个“删除标记”代替记录在偏移量表中的指针。给定一个磁盘块，如果刚开始块是空的，则几天后不能再向该块内插入记录？此时，该块内一共有多少条记录？

解答：

- 第一天：插入 4 条记录，800＋8＝808 B。
- 以后每天都增加：－400＋808＝408 B，注意删除标记的指针不可重用，因为之前可能有指针指向该记录。

$$808+408x <= 8192-8$$
$$x <= 18.1$$

因 x 为整数所以 x=18。当 x=18 时，即第 19 天插入记录后，占据的磁盘空间为：808+408*18=8152。此时共有记录数=4+18*2=40。

第 20 天，删除 2 条记录插入 2 条记录后就不能继续了……

2

- 假设我们用 SQL 基本表来存储文献:
  paper(id: int, title: varchar(200), abstract: varchar(1000))。

用户最常见的文献查询可以描述为“查询 abstract 中同时包含给定（多个）关键词的文献”。请回答下面问题：

1. 上述查询在 SQL 中该如何表示？请举例说明。
2. 我们想优化上述查询。假如在 abstract 上创建一个 B+-tree 索引，能不能提高此查询的效率（解释理由）？
3. 假设 DBMS 允许用户扩充新的索引结构。请给出一种优化上述查询的索引结构，并说明该索引为什么可以提高此查询性能。

答：

解答：

1. 用 Like 表示。例如要查询包含关键词 database 和 system 的文献:

```
Select * from paper
where abstract Like '%database%' and abstract Like '%system%'
```

2. 如果查询字段上建有 B+树索引，能否提高模糊查询效率与模糊查询的方式有关。分两种情况：

① LIKE a%：若 LIKE 中的查询条件是由常量字符串开头的，则 B+树索引有效。因为查询时可首先通过 B+树的根节点确定“a%”字符串所在的子树，并且根据前缀字符串可以继续往下比较，缩小搜索范围。因此，与没有B+树索引时相比，有 B+树索引时最后查询的范围可缩小到前缀字符串开头的键值集合，从而提高了查询效率。

② LIKE %a：若 LIKE 中的查询条件是由通配符开头的，则 B+树索引无助于查询效率的提高。因为即使有 B+树索引，查询时最坏情况下仍要搜索整个索引和整个记录集合，因此与没有索引时相比，查询效率不但没有提高，反而下降了，因为引入了索引查找的额外代价。

3. 可以通过扩充倒排索引来优化。倒排索引的基本结构：<keyword, page no>，当插入一个新的 paper 记录时，对其 abstract 字段进行分词，并将分词后的每个 keyword 加上该记录所在的 page no (如果包含该 keyword 的 paper 记录一个磁盘块存不下，则可以使用桶的首块号，即 bucket no) 插入到倒排索引中。执行多关键词查询时，首先查询倒排索引，然后获取每个关键词指向的 page no 或者 bucket no，然后只需要读入该 page 或者扫描该 bucket。如此一来，可以不需要扫描所有的 page 即可回答多关键词查询，所以该索引设计是有效的。

HW-3 (Query Optimization)

5. 4. 1

习题 5. 4. 1 下面是 4 个关系 W、X、Y、Z 的关键统计值：

$W(a,b)$	$X(b,c)$	$Y(c,d)$	$Z(d,e)$
$T(W)=400$	$T(X)=300$	$T(Y)=200$	$T(Z)=100$
$V(W,a)=50$	$V(X,b)=60$	$V(Y,c)=50$	$V(Z,d)=10$
$V(W,b)=40$	$V(X,c)=100$	$V(Y,d)=20$	$V(Z,e)=50$

估计下列表达式结果关系的大小：

a) $W \bowtie X \bowtie Y \bowtie Z$　b) $\sigma_{a=10}(W)$　c) $\sigma_{c=20}(Y)$
d) $\sigma_{c=20}(Y) \bowtie Z$　e) $W \times Y$　f) $\sigma_{d>10}(Z)$
g) $\sigma_{a=1 \text{ AND } b=2}(W)$　h) $\sigma_{a=1 \text{ AND } b>2}(W)$　i) $X \bowtie_{X.c<Y.c} Y$

a)

由定义

$$\text{若 W} = \text{R1} \bowtie \text{R2}\left(\text{共同属性是 A}\right)$$

$$\text{则 T(W)} = \frac{T(R1)\cdot T(R2)}{\max\{V(R1,A),V(R2,A)\}}$$

得

$$\frac{T(W)\cdot T(X)\cdot T(Y)\cdot T(Z)}{V(X,b)\cdot V(X,c)\cdot V(Y,d)} = \frac{400\times 300\times 200\times 100}{60\times 100\times 20} = 20000$$

b)

$$\frac{T(W)}{V(W,a)} = \frac{400}{50} = 8$$

c)

$$\frac{T(Y)}{V(Y,c)} = \frac{200}{50} = 4$$

d)

$\sigma_{c=20}(Y) \bowtie Z$

因为 $V(Z,d) > T(\sigma_{c=20}(Y))$，所以

$$\frac{T(\sigma_{c=20}(Y)) * T(Z)}{V(Z,d)} = \frac{4 * 100}{10} = 40$$

e)

$$W \times Y = T(W) \times T(Y) = 400 \times 200 = 80000$$

f)

$$\frac{T(Z)}{3} = \frac{100}{3} = 33.3$$

g)

$$\frac{T(W)}{V(W,a) \cdot V(W,b)} = \frac{400}{50 \times 40} = 0.2$$

h)

$$\frac{T(W)}{V(W,a) \times 3} = \frac{400}{50 \times 3} = 2.67$$

i)

$$\frac{T(X) \cdot T(Y)}{3} = \frac{300 \times 200}{3} = 20000$$

HW-4 (Join Algorithms)

4. 3. 5

!习题 4. 3. 5 如果 R 和 S 都是非聚集的，似乎嵌套循环连接将需要大约 $T(R)T(S)/M$ 次磁盘 I/O 时间。

a) 你怎样做才能明显好于这个代价？

b) 如果 R 和 S 中只有一个是非聚集的，你怎样执行嵌套循环连接？考虑两种情况：较大的关系是非聚集的和较小的是非聚集的。

a)

假设 T(R)>T(S)，S(S)=S(R)，即 B(R)>B(S)。应该先从 S 中取 M-1 块放入内存，和 R 中所有元素连接，然后循环这个过程。

这样，总 IO 为

$$\frac{B(S)}{M-1}\cdot\left(\frac{M-1}{S(S)}+T(R)\right)$$

又因为

$$S(S)=\frac{B(S)}{T(S)}$$

所以上式为

$$\frac{B(S)}{M-1}\cdot\left(\frac{M-1}{\frac{B(S)}{T(S)}}+T(R)\right)=\mathrm{T(S)}+\frac{B(S)\cdot T(R)}{M-1}$$

当 R 比 S 大很多，R、S 较大时，$B(S)\cdot T(R)\gg T(S)$

上式约等于

$$\frac{B(S)\cdot T(R)}{M}$$

和题目中的$\frac{T(S)\cdot T(R)}{M}$相比，假设一个块可以存 n 行数据，IO 就是原来的$\frac{1}{n}$倍。

方法 2

每次读入R时，将R填满M-1个块以减少嵌套循环次数
假设S(R)=S(S),则外层循环次数变为T(R)*S(R)/(M-1)
每次循环需要的I/O次数为(M-1)/S(R)+T(S)
总代价为T(R)+T(S)*T(R)*S(R)/(M-1)

方法 3

先聚集，再连接
聚集T(S)+T(R) + B(S)+B(R)
连接B(S)B(R)/M

T(S)+T(R)+B(S)+B(R)+B(S)B(R)/M

b)

R 是聚集的

方案 1 用 M-1 块读 S

$$\frac{B(S)}{M-1}\cdot\left(\frac{M-1}{\frac{B(S)}{T(S)}}+B(R)\right)=\mathrm{T(S)}+\frac{B(S)\cdot B(R)}{M-1}$$

方案 2 用 M-1 块读 R

$$\frac{B(R)}{M-1}\cdot\big(M-1+T(S)\big)=B(R)+\frac{B(R)\cdot T(S)}{M-1}$$

在一般情况下，R、S 较大，方案 1 更好。

S 是聚集的

方案 1 用 M-1 块读 S

$$\frac{B(S)}{M-1}\cdot\big(M-1+T(R)\big)=\mathrm{B(S)}+\frac{B(S)\cdot T(R)}{M-1}$$

方案 2 用 M-1 块读 R

$$\frac{B(R)}{M-1}\cdot\left(\frac{M-1}{\frac{B(R)}{T(R)}}+B(S)\right)=\mathrm{T(R)}+\frac{B(S)\cdot B(R)}{M-1}$$

在一般情况下，R、S 较大，方案 2 更好。

综上&总结

假如要直接做决定，应该用 M-1 块读非聚集的关系到内存。
假如已知统计量，可以计算两个方案的结果，选代价小的执行。

b

	R<S	
	先R	先S
R聚集	B(R)+B(R)*T(S)/(M-1)	T(S)+B(R)*T(S)*S(S)/(M-1)
S聚集	T(R)+B(S)*T(R)*S(R)/(M-1)	B(S)+T(R)*B(S)/(M-1)
均不聚集	T(R)+T(S)*T(R)*S(R)/(M-1)	T(S)+T(R)*T(S)*S(S)/(M-1)

聚集：B=TS

4. 4. 5

!习题 4. 4. 5 假设这节中所描述算法的第二趟不需要所有的 M 个缓冲区，因为子表数小于 M。我们怎样通过使用额外的缓冲区来节省磁盘 I/O?

第一趟无需将排序完毕的 chunk 全部写回磁盘，可以留在内存中。第二趟这一部分直接在内存中进行归并排序，速度更快。而且第二趟排序完也可以留在内存中，之后进行操作又可以减少一些 IO 操作。

原本我们需要将第一趟中得到的有序子表都写回磁盘，现在由于子表数小于M，可以将部分子表不写回，直接存储在内存缓冲区中，从而减少第二趟中的读子表操作，对于这样每块我们节省了 2 次IO.

HW-5 (Log & Recovery)

下面是一个数据库系统开始运行后的日志记录，……

1、下面是一个数据库系统开始运行后的日志记录，该数据库系统支持 simple checkpoint。

1) <T1, Begin Transaction>
2) <T1, A, 49, 20>
3) <T2, Begin Transaction>
4) <T1, B, 250, 20>
5) <T1, A, 75, 49>
6) <T2, C, 35, 20>
7) <T2, D, 45, 20>
8) <T1, Commit Transaction>
9) <T3, Begin Transaction>
10) <T3, E, 55, 20>
---------- ①
11) <T2, D, 46, 45>
12) <T2, C, 65, 35>
13) <T2, Commit Transaction>
---------- ②
14) <T3, Commit Transaction>
15) <CHECKPOINT>
16) <T4, Begin Transaction>
17) <T4, F, 100, 20>
18) <T4, G, 111, 20>
---------- ③
19) <T4, F, 150, 100>
20) <T4, Commit Transaction>

设日志修改记录的格式为 <Tid, Variable, New value, Old value>，请给出对于题中所示①、②、③三种故障情形下，数据库系统恢复的过程以及数据元素 A, B, C, D, E, F 和 G 在执行了恢复过程后的值。

此题疑点

助教答案 Redo 完要在日志写入 Commit，但课件和参考资料没此类说法。不要写入 Commit 的可能性更高

①

要 Undo T2、T3，Redo T1。

先 Undo：<T3, E, 20> <T2, D, 20> <T2, C, 20> <Abort, T2> <Abort, T3>

再 Redo：<T1, A, 49> <T1, B, 250> <T1, A, 75> <Commit, T1>

综上 A=75, B=250, C=20, D=20, E=20 F=20, G=20

②

要 Undo T3，Redo T1、T2。

先 Undo：<T3, E, 20> <abort, T3>

再 Redo：<T1, A, 49> <T1, B, 250> <T1, A, 75> <T2, C, 35> <T2, D, 45> <T2, D, 46> <T2, C, 65> <Commit, T1> <Commit, T2>

综上 A=75, B=250, C=65, D=46, E=20 F=20, G=20

③

T1、T2、T3 已经 Commit，无需 Undo/Redo。

要 Undo T4。

Undo：<T4, G, 20> <T4, F, 20> <abort, T4>

综上 A=75, B=250, C=65, D=46, E=55, F=20, G=20

检查点之前：
A=75,B=250,C=65,D=46
E=55,F=20,G=20
检查点之后，T4：Undo
F=20，G=20
T4:abort

HW-6 (Concurrency Control)

1、判断下面的并发调度是否冲突可串？如果是，请给出冲突等价的串行事务顺序；如果不是，请解释理由

w4(D); r1(A); w2(A); r4(A); r1(C); r2(B); w2(B); r3(B); r3(A); w2(D); w3(B); r4(B); w3(C); r4(C); w4(B)

不是冲突可串的。

因为用优先图判断冲突可串时，w4(D)、w2(D)和 w2(A)、r4(A)会导致 2 和 4 之间有回路，所以这个并发调度不是冲突可串的。

WW、WR、RW 这些冲突操作决定事务的序

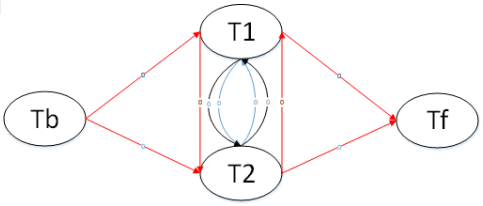
2

2、对于课堂讲义上提及的 Schedule D，请画出该调度对应的 Polygraph。假设现在我们用基于 S、X 和 U 锁的 2PL 来实现并发控制，请给出锁管理器加入了(s/x/u)Lock 和 Unlock 操作后的调度序列，要求用 r、w 序列的形式表示最后的调度。

- WB(B)Wb(A) R1(A)W1(A)R2(A)W2(A)R2(B)W2(B)R1(B)W1(B) Rf(B)Rf(A)
- 解读：对每个读操作，寻找相同元素的前一个写操作
  - (3a) If wi(A) => rj(A) in S, add Ti → Tj(0边)
  - (3b) For each wi(A) =>rj(A) do
    - k Tb和Tf
    - P边,$k \neq i, j, b, f$ 0条
    - 0边
      - i = b
        - j = 1 k = 2
        - j = 2 k = 1
      - j = f
        - i = 1 k = 2
        - I = 2 k = 1

Schedule D

T1	T2	A	B
Read(A, t); t ← t+100		25	25
Write(A, t);		125	25
	Read(A, s); s ← s×2;		
	Write(A, s);	250	25
	Read(B, s); s ← s×2;		
	Write(B, s);	250	50
Read(B, t);			
t ← t+100;			
Write(B, t);		250	150



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

• UL1(A)	
• R1(A)	• UL2(A)
• Upgrade(A)	• Wait
• W1(A)	• Wait
• UL1(B)	• Wait
• U1(A)	• Wait
• R1(B)	• R2(A)
• Upgrade(B)	• Upgrade(A)
• W1(B)	• W2(A)
• U1(B)	• UL2(B)
	• U2(A)
	• R2(B)
	• Upgrade(B)
	• W2(B)
	• U2(B)

- Update Lock
  - 如果事务取得了数据R上的更新锁，则可以读R，并且可以在以后升级为X锁
  - 单纯的S锁不能升级为X锁
  - 如果事务持有了R上的Update Lock，则其它事务不能得到R上的S锁、X锁以及Update锁
  - 如果事务持有了R上的S Lock，则其它事务可以获取R上的Update Lock
    - <S,U>相容
    - <U,S>不相容
- Update Lock主要防止由于SX锁引起的死锁问题
  - Update Lock并不能保证不会产生死锁，只是针对共享锁和排他锁提出了一种较为简单的解决方式
  - T1 R(A)W(A)
  - T2 R(A)W(A)
- 不能解决的问题
  - T1 W(A)W(B)
  - T2 W(B)W(A)
- 排他锁和update锁的区别
  - T1持有S锁，T2不能申请X锁，但是可以申请U锁，T2持有U锁后，其他事务便不能再持有S锁或X锁

最后的调度 r、w 序列：

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

3 如果并发调度所有事务遵守两阶段锁协议，会发生丢失更新、脏读问题吗？

不会丢失更新，可能会脏读。
老师说过：脏读问题确切的定义是“未提交依赖问题”，即一个事务读或写的数据是另一个事务未提交的数据。丢失更新的定义是“两个事务独立地更新同一个数据（默认是一个元组），其中某一个事务的更新被另一个事务的更新所覆盖导致更新丢失”。2PL 可以避免出现丢失更新，但是不能避免脏读以及不一致分析。实际 DBMS 实现 2PL 时默认用严格两阶段锁 SS2PL（所有锁保持到事务结束 strong strict 2 phase locking）。具体如何加锁取决于 DBMS 支持什么样的事务隔离级别。