MySQL锁技能进阶(一)

根据加锁的范围，MySQL 锁大致可以分成全局锁、表级锁和行锁三类

MySQL全局锁： flush tables with read lock (FTWRL)

set global readonly=true 思考此操作有何风险？

MySQL表级锁有两种：一种是表锁，一种是元数据锁（meta data lock，MDL)。

表锁：lock tables … read/write

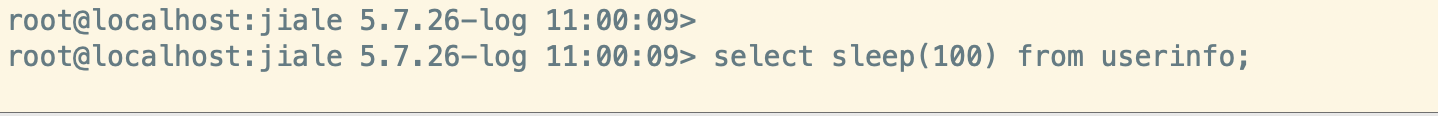
元数据锁：MDL（metadata lock)

MDL 不需要显式使用，在访问一个表的时候会被自动加上。

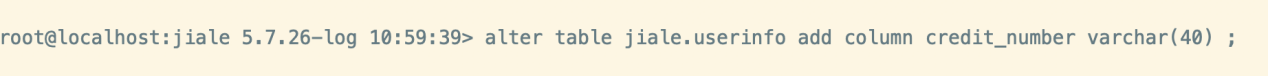
MDL 的作用是，保证读写的正确性。你可以想象一下，如果一个查询正在遍历一个表中的数据，而执行期间另一个线程对这个表结构做变更，删除/增加一列，那么查询线程拿到的结果跟表结构对不上，肯定是不行的。

模拟：

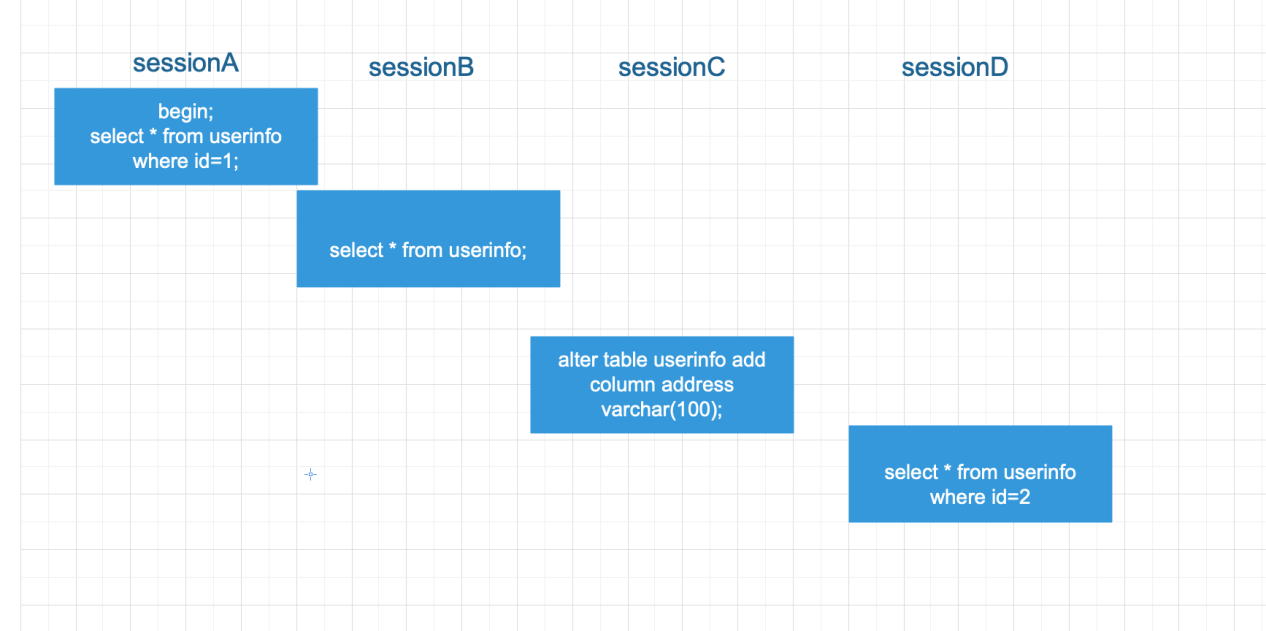
session A:



session B:



长事务引发的现网事故。



思考如何规避风险？

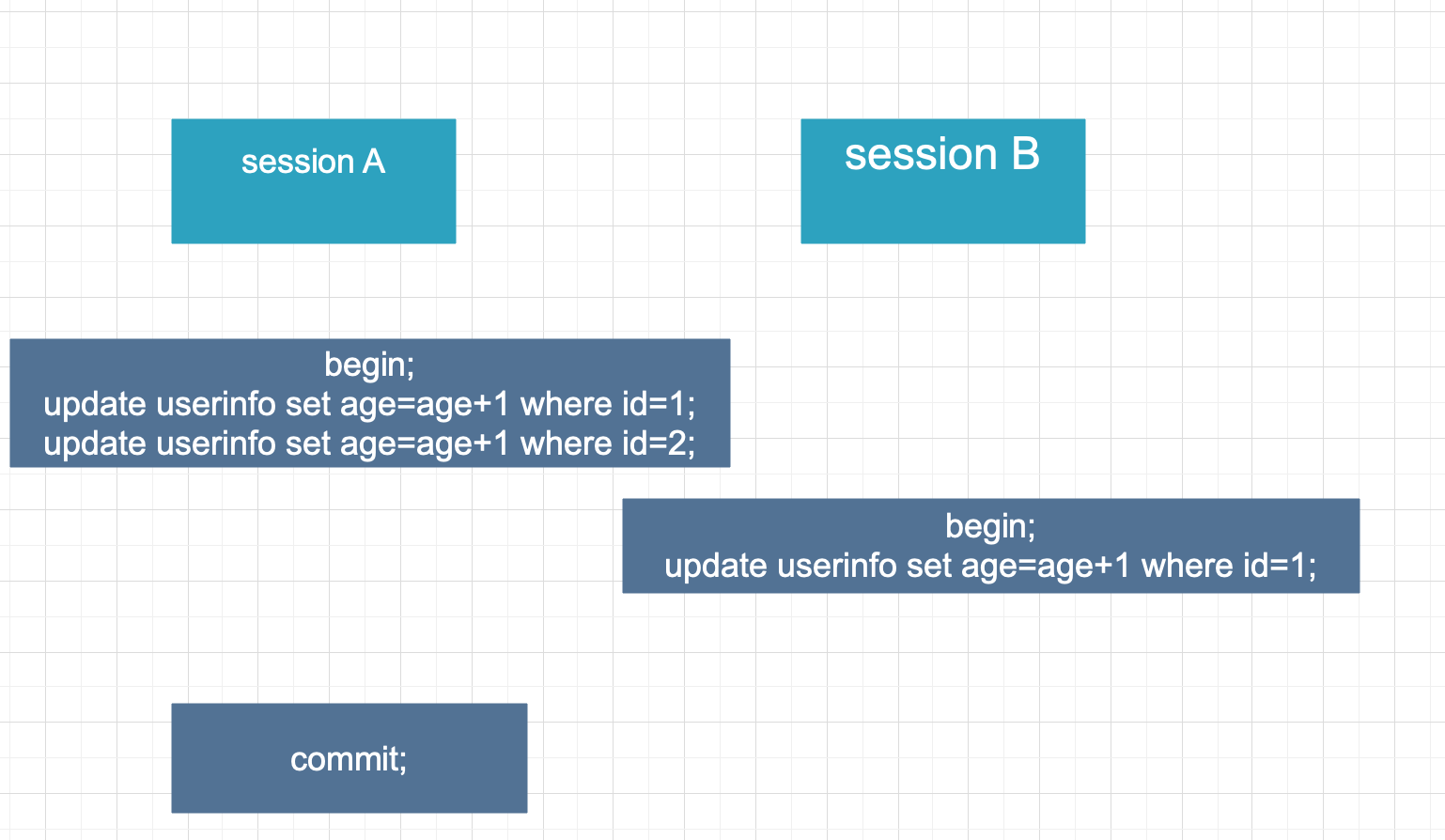
1. 代码/程序设计开启事务记得及时commit。
2. 上线前开启general\_log .观察是否有未commit事务等引起的异常。
3. 定期检测并KILL掉长时间运行线程。（如pt-kill）但存在误杀的风险。
4. 使用mariadb版本自带 DDL nowait/wait

alter table tbname nowait add/drop column ...

alter table tbname wait N add/drop column ...

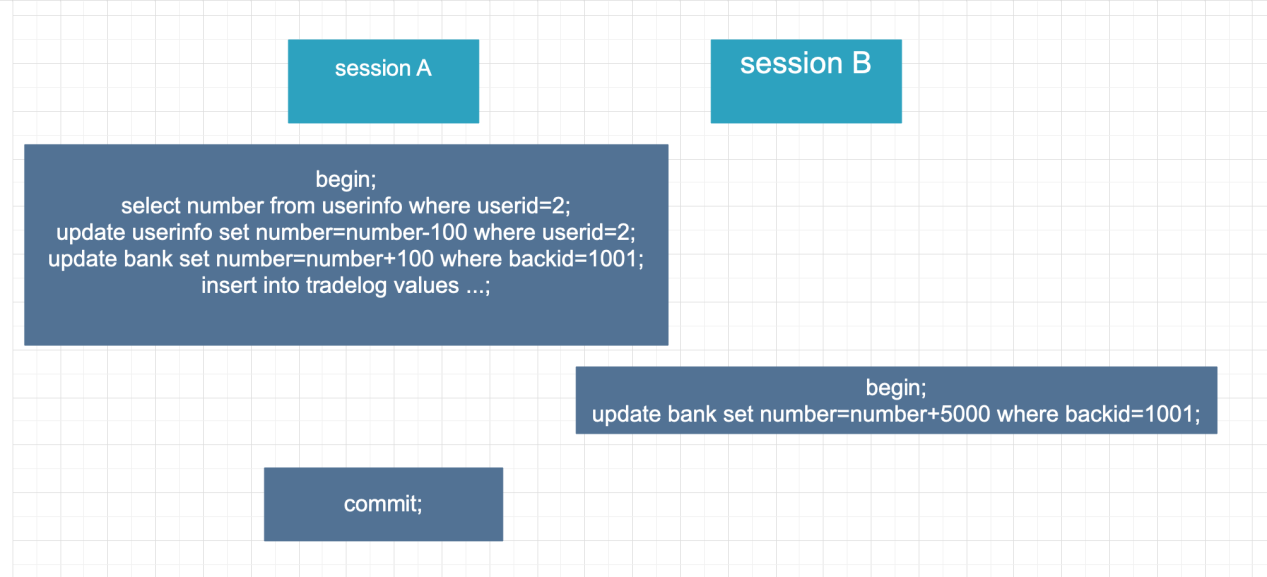
MySQL行锁：MySQL 的行锁是在引擎层由各个引擎自己实现 。（INNODB取代MyISAM的重要原因。）

两阶段锁协议:



结论：在 InnoDB 事务中，行锁是在需要的时候才加上的，但并不是不需要了就立刻释放，而是要等到事务提交时才释放。这就是两阶段锁协议（加锁/解锁）

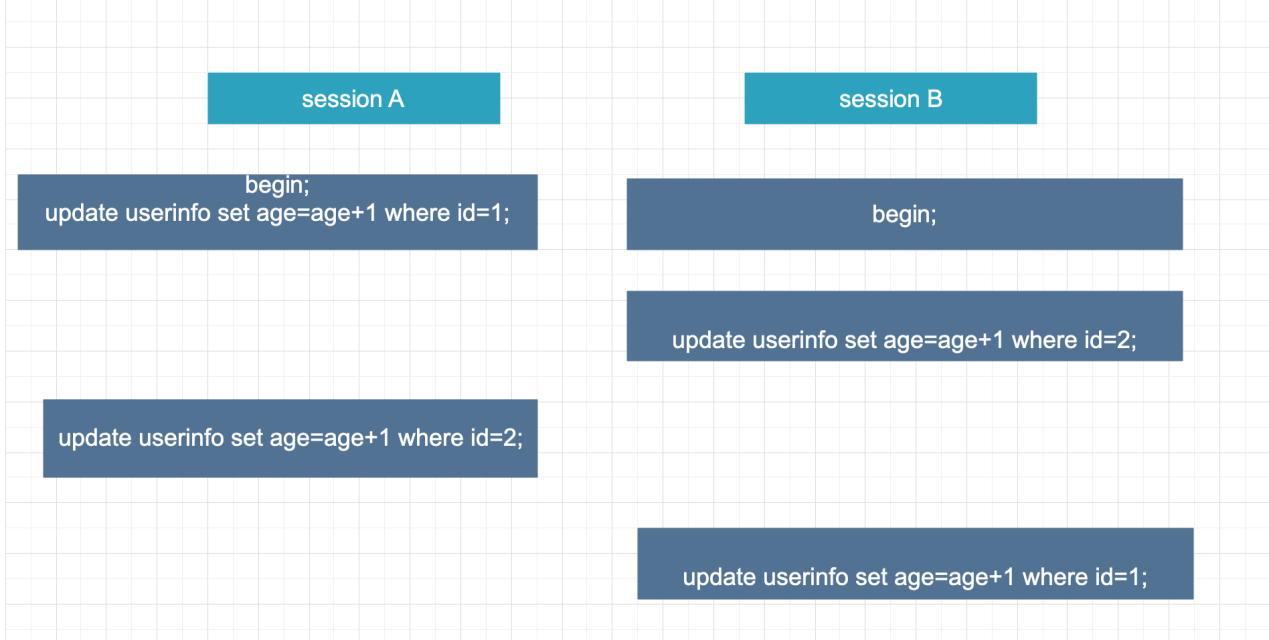
由此给我们的思考？



如果事务中需要锁定多行，把最可能造成锁冲突，最影响并发量的锁放在最后。提高并发性能手段之一。

大家在工作中一定很关心并发处理问题。在数据库中使用多版本并发控制（MVCC）来解决提升应用并发量（不在本话题内）。使用死锁检测来解决死锁问题。

那么死锁现象和死锁检测，以及如何尽量规避死锁呢？



MySQL 通过innodb\_lock\_wait\_timeout解决锁超时。默认50秒

MySQL 通过innodb\_deadlock\_detect =on 开启死锁检测。在死锁发生时快速发现并处理。

但是开启死锁检测成本很高。

原因分析：每个新来的被堵住的线程，都要判断会不会由于自己的加入导致死锁，这是一个时间复杂度为 O(n) 的操作。假设有 1000 个并发线程要同时更新同一行，那么死锁检测操作就是 100 万这个量级。虽然最终检测的结果是没有死锁，但是这期间要消耗大量的 CPU 资源。对于开启或关闭死锁检测需要权衡利弊。

关注点：死锁对程序的影响。死锁检测对性能的损耗（会出一期关于mysql参数对性能影响专题）

那么如何解决呢？

控制并发度

比如同一行同时最多只有 10 个线程在更新，那么死锁检测的成本很低，就不会出现这个问题。一个直接的想法就是，在客户端做并发控制。但是，你会很快发现这个方法不太可行，因为客户端很多。假设有 600 个客户端，这样即使每个客户端控制到只有 5 个并发线程，汇总到数据库服务端以后，峰值并发数也可能要达到 3000。

可以考虑使用中间件或修改源码（对于相同行的更新，在进入引擎层前排队）实现。

有RDS团队在服务层实现 如阿里RDS。通过监控系统status变量threads\_running，当满足拒绝条件时，拒绝执行sql，返回用户：MySQL Server is too busy，判断逻辑在dispatch\_command中，sql解析之后。

增加的系统variables：

1. threads\_running\_ctl\_mode： 限流的sql类型，有两个取值：[ALL | SELECTS]，默认SELECTS，设置为ALL需谨慎。

2.threads\_running\_high\_watermark： 限流水位值，只有threads\_running超过此值才会触发，默认值为max\_connections，当set global threads\_running\_high\_watermark=0时自动设置为max\_connections。

拒绝执行的必要条件：

     1..threads\_running超过threads\_running\_high\_watermark。

     2..threads\_running\_ctl\_mode与sql类型相符。

设计上优化并发

把对单行锁竞争改成逻辑上多行锁竞争，减少行争用的频率和概率，一定程度上可减少锁冲突，从而提升并发性能。

案例：

影院售票系统开放会员免费活动：

伪代码逻辑

begin;

select ticket from sales where sales\_filmid=1;

if ticket>0 ;then

update sales set ticket=ticket-1 where sales\_filmid=1; //ticket=1000

commit;

改造后伪代码逻辑

begin;

point=random.randint((1,10))

select ticket from sales where sales\_filmid=1 and sub=$point

if ticket>0 ;then

update sales set ticket=ticket-1 where sales\_filmid=1 and sub=$point; // ticket=100

commit;

思考：此方案存在误判风险。需要在设计机制上加更多的判断，以防止出现超卖/少卖等此类数据不一致可能。

我将会在锁技能进阶（二）补充关于锁流程，定位锁相关技能。

MySQL锁技能进阶二

锁机制：

数据库区别于文件系统的一个关键特性之一。用于管理对共享资源的并发访问。

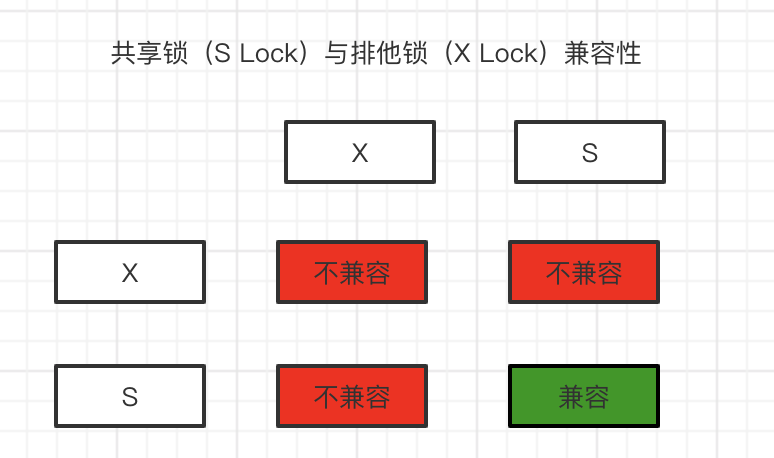
各种关系型数据库类似，但锁实现方式上有各自的特点。

MySQL InnoDB 存储引擎,锁设计粒度:行锁。

MySQL MyISAM 存储引擎,锁设计粒度:表锁。

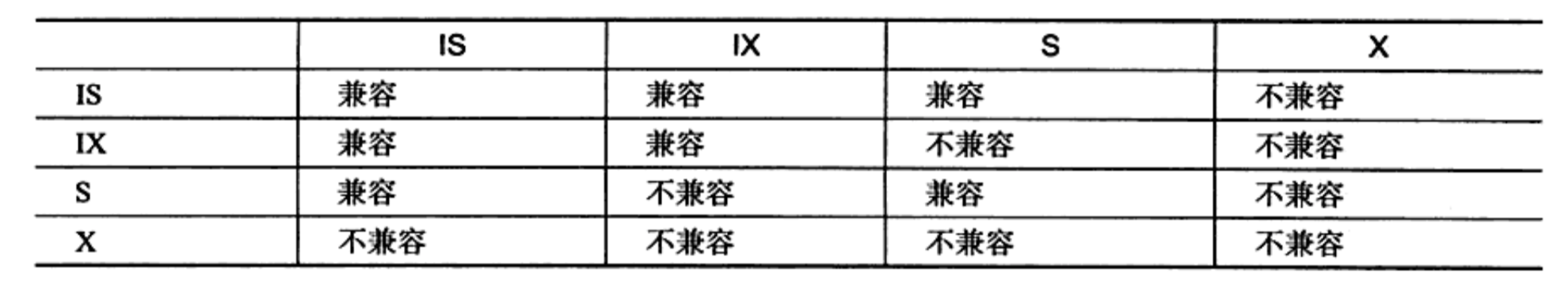
其他数据库如SQL server 锁设计粒度：页锁。热点数据页并发访问。后通过乐观/悲观行锁控制并发访问（但会造成锁升级）

lock对象是事务，用来锁定的是数据库中的对象：表，页，行。并且lock对象仅在事务commit或rollback后进行释放。（不同隔离级别释放的时间可能不同）

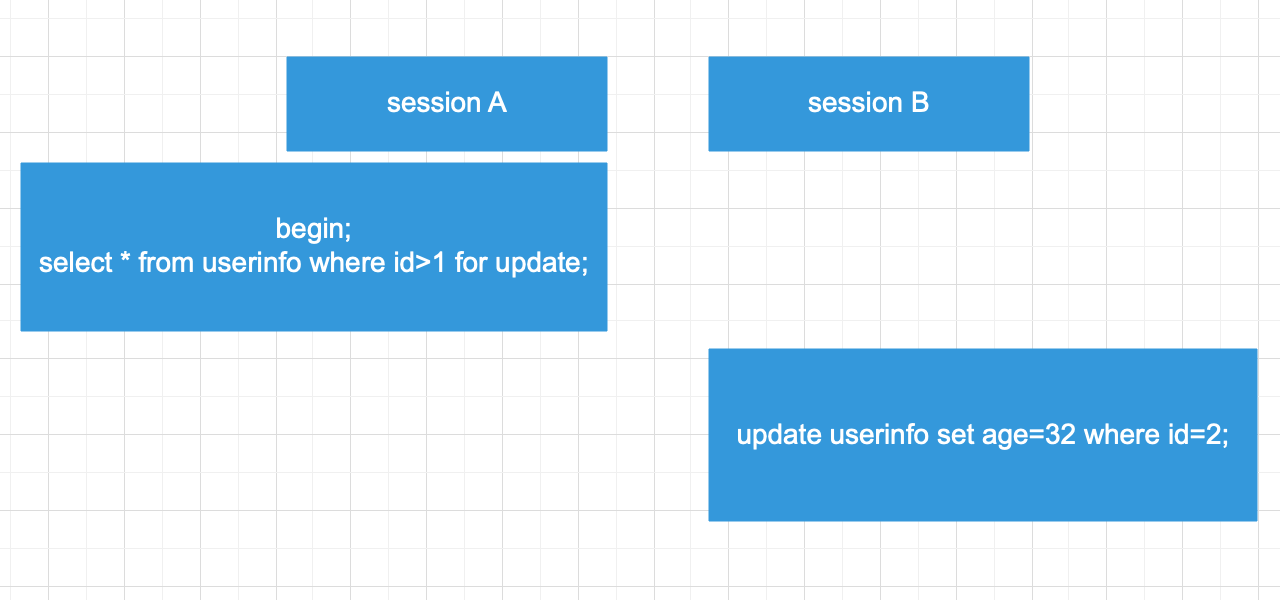


InnoDB支持多粒度锁定。允许事务在行级和表级上的锁同时存在。为了支持在不同粒度上加锁操作 >>> 意向锁(Intention Lock)

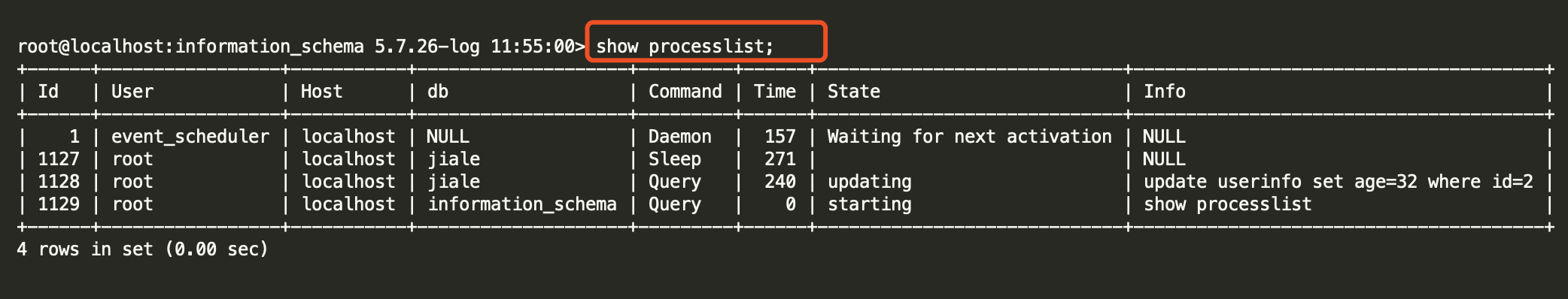
兼容矩阵：



模拟加锁：



-- 查看数据库锁请求情况



root@localhost:(none) 5.7.26-log 11:51:27> show engine innodb status\G （省略与本文无关输出信息）

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 1. row \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Type: InnoDB

Name:

Status:

=====================================

2019-12-09 11:52:42 0x7f6d5d1c8700 INNODB MONITOR OUTPUT

=====================================

Per second averages calculated from the last 14 seconds

.......

------------

TRANSACTIONS

------------

Trx id counter 1593

Purge done for trx's n:o < 1590 undo n:o < 0 state: running but idle

History list length 8

LIST OF TRANSACTIONS FOR EACH SESSION:

---TRANSACTION 421582651166320, not started

0 lock struct(s), heap size 1136, 0 row lock(s)

---TRANSACTION 421582651165408, not started

0 lock struct(s), heap size 1136, 0 row lock(s)

---TRANSACTION 1592, ACTIVE 5 sec starting index read

mysql tables in use 1, locked 1

LOCK WAIT 2 lock struct(s), heap size 1136, 1 row lock(s)

MySQL thread id 1128, OS thread handle 140107690583808, query id 2591 localhost root updating

update userinfo set age=32 where id=2

------- TRX HAS BEEN WAITING 5 SEC FOR THIS LOCK TO BE GRANTED:

RECORD LOCKS space id 37 page no 3 n bits 80 index PRIMARY of table `jiale`.`userinfo` trx id 1592 lock\_mode X locks rec but not gap waiting

Record lock, heap no 3 PHYSICAL RECORD: n\_fields 7; compact format; info bits 0

0: len 4; hex 80000002; asc ;;

1: len 6; hex 00000000062d; asc -;;

2: len 7; hex d400000187011c; asc ;;

3: len 8; hex 800000000000000b; asc ;;

4: len 7; hex 6368656e6c6569; asc chenlei;;

5: len 1; hex a8; asc ;;

6: len 1; hex 80; asc ;;

------------------

---TRANSACTION 1591, ACTIVE 36 sec

2 lock struct(s), heap size 1136, 9 row lock(s)

MySQL thread id 1127, OS thread handle 140106502825728, query id 2589 localhost root

--------

.......

tips: innodb\_status\_output\_locks=on ,show engine innodb status更多信息

root@localhost:information\_schema 5.7.26-log 11:54:25> select \* from INNODB\_TRX\G

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 1. row \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

trx\_id: 1592

trx\_state: LOCK WAIT

trx\_started: 2019-12-09 11:52:37

trx\_requested\_lock\_id: 1592:37:3:3

trx\_wait\_started: 2019-12-09 11:52:37

trx\_weight: 2

trx\_mysql\_thread\_id: 1128

trx\_query: update userinfo set age=32 where id=2

trx\_operation\_state: starting index read

trx\_tables\_in\_use: 1

trx\_tables\_locked: 1

trx\_lock\_structs: 2

trx\_lock\_memory\_bytes: 1136

trx\_rows\_locked: 1

trx\_rows\_modified: 0

trx\_concurrency\_tickets: 0

trx\_isolation\_level: REPEATABLE READ

trx\_unique\_checks: 1

trx\_foreign\_key\_checks: 1

trx\_last\_foreign\_key\_error: NULL

trx\_adaptive\_hash\_latched: 0

trx\_adaptive\_hash\_timeout: 0

trx\_is\_read\_only: 0

trx\_autocommit\_non\_locking: 0

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 2. row \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

trx\_id: 1591

trx\_state: RUNNING

trx\_started: 2019-12-09 11:52:06

trx\_requested\_lock\_id: NULL

trx\_wait\_started: NULL

trx\_weight: 2

trx\_mysql\_thread\_id: 1127

trx\_query: NULL

trx\_operation\_state: NULL

trx\_tables\_in\_use: 0

trx\_tables\_locked: 1

trx\_lock\_structs: 2

trx\_lock\_memory\_bytes: 1136

trx\_rows\_locked: 9

trx\_rows\_modified: 0

trx\_concurrency\_tickets: 0

trx\_isolation\_level: REPEATABLE READ

trx\_unique\_checks: 1

trx\_foreign\_key\_checks: 1

trx\_last\_foreign\_key\_error: NULL

trx\_adaptive\_hash\_latched: 0

trx\_adaptive\_hash\_timeout: 0

trx\_is\_read\_only: 0

trx\_autocommit\_non\_locking: 0

2 rows in set (0.00 sec)

root@localhost:information\_schema 5.7.26-log 11:54:29> select \* from INNODB\_LOCKS;

+-------------+-------------+-----------+-----------+--------------------+------------+------------+-----------+----------+-----------+

| lock\_id | lock\_trx\_id | lock\_mode | lock\_type | lock\_table | lock\_index | lock\_space | lock\_page | lock\_rec | lock\_data |

+-------------+-------------+-----------+-----------+--------------------+------------+------------+-----------+----------+-----------+

| 1592:37:3:3 | 1592 | X | RECORD | `jiale`.`userinfo` | PRIMARY | 37 | 3 | 3 | 2 |

| 1591:37:3:3 | 1591 | X | RECORD | `jiale`.`userinfo` | PRIMARY | 37 | 3 | 3 | 2 |

+-------------+-------------+-----------+-----------+--------------------+------------+------------+-----------+----------+-----------+

2 rows in set, 1 warning (0.00 sec)

Warning (Code 1681): 'INFORMATION\_SCHEMA.INNODB\_LOCKS' is deprecated and will be removed in a future release.

root@localhost:information\_schema 5.7.26-log 11:54:43> select \* from INNODB\_LOCK\_WAITS;

+-------------------+-------------------+-----------------+------------------+

| requesting\_trx\_id | requested\_lock\_id | blocking\_trx\_id | blocking\_lock\_id |

+-------------------+-------------------+-----------------+------------------+

| 1592 | 1592:37:3:3 | 1591 | 1591:37:3:3 |

+-------------------+-------------------+-----------------+------------------+

1 row in set, 1 warning (0.00 sec)

tips:可通过events\_statements\_current查看持有锁SQL。

root@localhost:performance\_schema 5.7.26-log 12:00:20> select \* from performance\_schema.events\_statements\_current where THREAD\_ID=1127\G

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 1. row \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

THREAD\_ID: 1127

EVENT\_ID: 9

END\_EVENT\_ID: 9

EVENT\_NAME: statement/sql/select

SOURCE:

TIMER\_START: 2080305515241809000

TIMER\_END: 2080305515906717000

TIMER\_WAIT: 664908000

LOCK\_TIME: 293000000

SQL\_TEXT: select \* from userinfo where id>1 for update

DIGEST: bcac85243232e159f51f6e5604c95b20

DIGEST\_TEXT: SELECT \* FROM `userinfo` WHERE `id` > ? FOR UPDATE

CURRENT\_SCHEMA: jiale

OBJECT\_TYPE: NULL

OBJECT\_SCHEMA: NULL

OBJECT\_NAME: NULL

OBJECT\_INSTANCE\_BEGIN: NULL

MYSQL\_ERRNO: 0

RETURNED\_SQLSTATE: NULL

MESSAGE\_TEXT: NULL

ERRORS: 0

WARNINGS: 0

ROWS\_AFFECTED: 0

ROWS\_SENT: 8

ROWS\_EXAMINED: 8

CREATED\_TMP\_DISK\_TABLES: 0

CREATED\_TMP\_TABLES: 0

SELECT\_FULL\_JOIN: 0

SELECT\_FULL\_RANGE\_JOIN: 0

SELECT\_RANGE: 1

SELECT\_RANGE\_CHECK: 0

SELECT\_SCAN: 0

SORT\_MERGE\_PASSES: 0

SORT\_RANGE: 0

SORT\_ROWS: 0

SORT\_SCAN: 0

NO\_INDEX\_USED: 0

NO\_GOOD\_INDEX\_USED: 0

NESTING\_EVENT\_ID: NULL

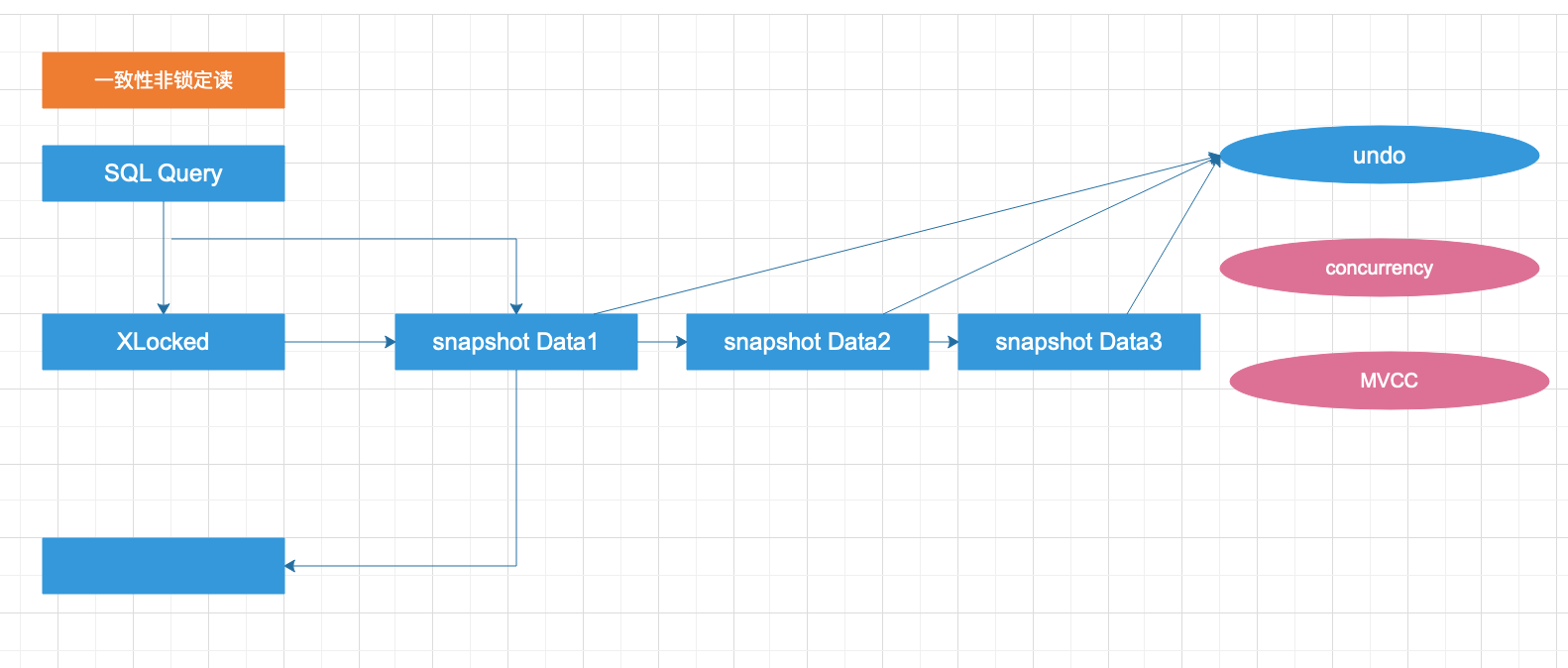
NESTING\_EVENT\_TYPE: NULL

NESTING\_EVENT\_LEVEL: 0

1 row in set (0.00 sec)

了解两个概念：一致性非锁定读/一致性锁定读

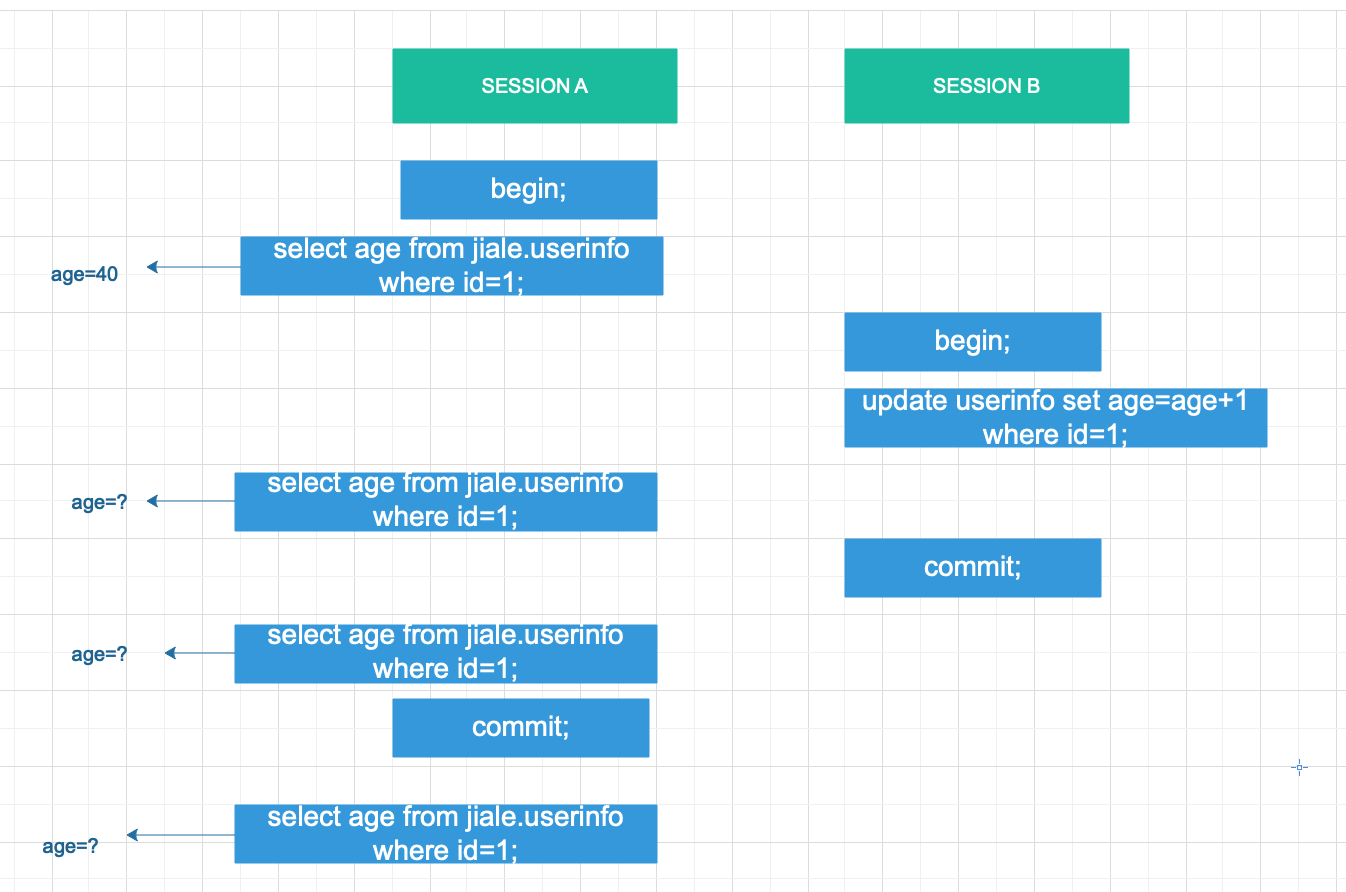
-- 一致性非锁定读（快照读）



在事务隔离级别READ COMMITTED（RC），REPEATABLE READ（RR）下，InnoDB存储引擎使用非锁定一致性读。但是对于快照数据定义却不相同。

RC：非一致性读总是读取被锁定行的最新（一份快照）数据。

RR：非一致性读总是读取事务开始时行版本数据。



-- 一致性锁定读（locking read）

select ... for update;

select ... lock in share mode;

-- 一致性锁定读加锁方式：

select ... for update; 对读取行记录➕ X锁，其他事务不能对已锁定行加任何锁。

select ... lock in share mode; 对读取行记录➕S锁，其他事务可以对已锁定行加S锁，但是如果加X锁会被阻塞。

行锁算法：

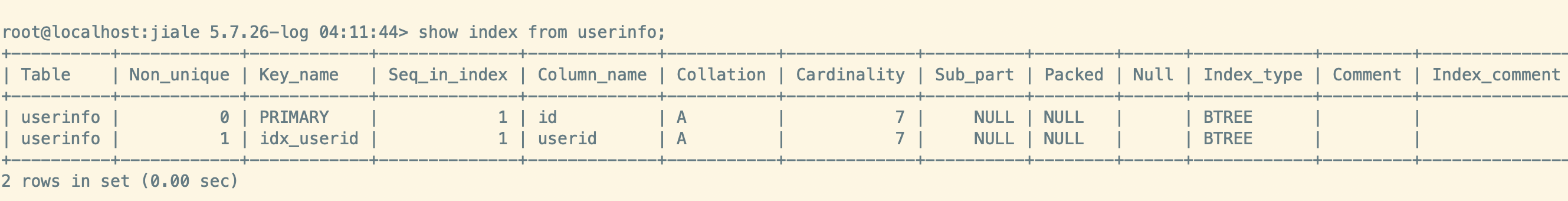
record lock:单个行记录上的锁

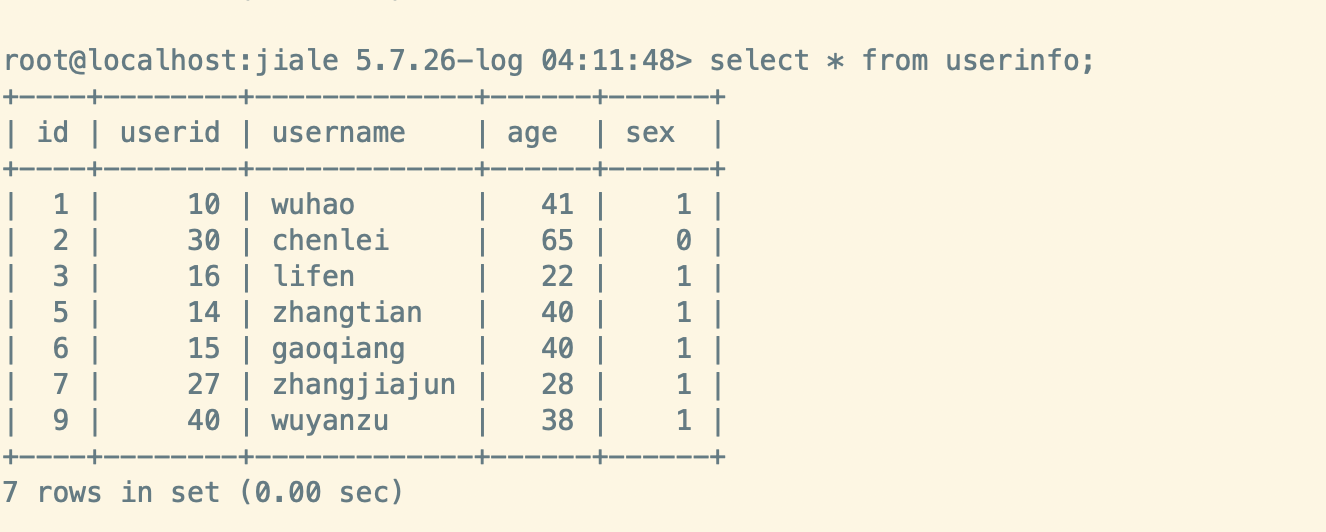
gap lock:间隙锁，锁定的一个范围，但不包括记录本身

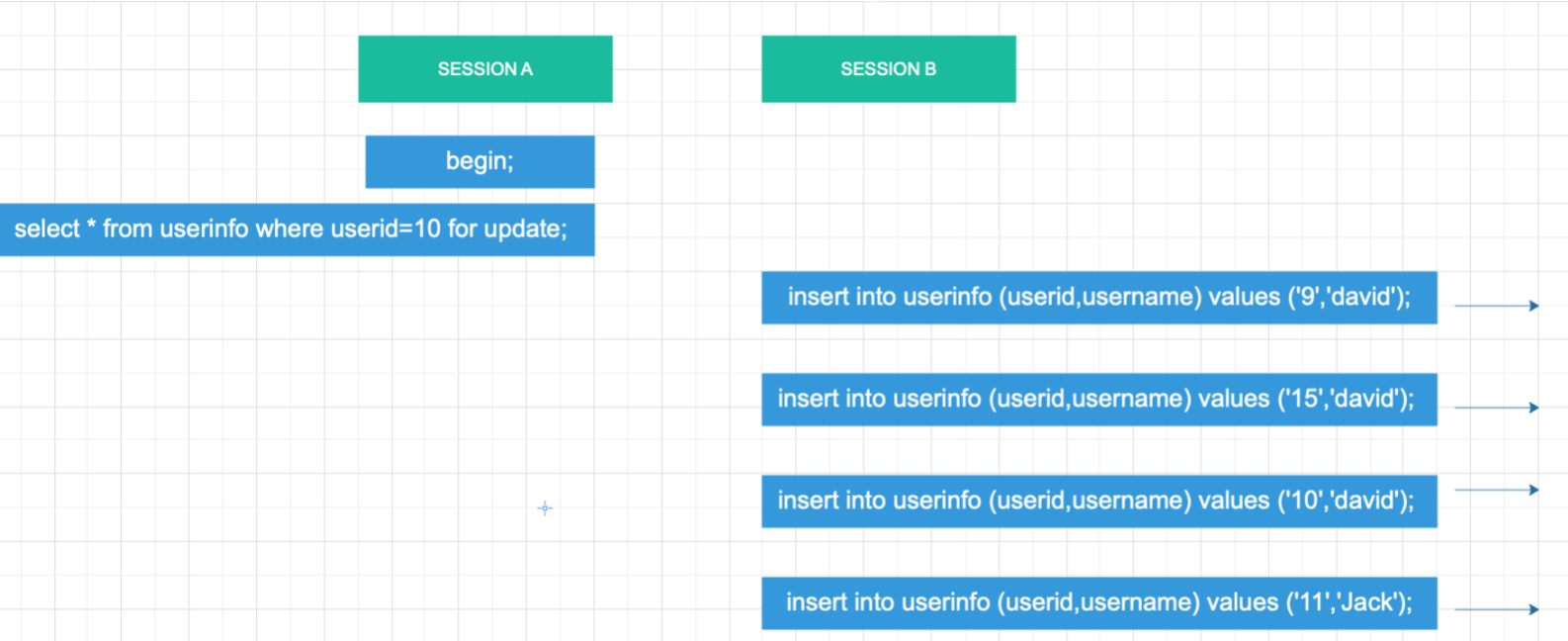
next-key lock:gap lock + record lock ，锁定一个范围，并且锁定记录本身（RR模式下防止幻读）

重点：查询条件含有唯一属性时，InnoDB（优化锁）next-key lock >>>record lock

模拟加锁，思考事务运行状态：







如何查看加锁过程：

MySQL-8.0新增performance\_schema.data\_locks表。

通过查看此表可清晰知道MySQL的加锁过程。再结合兼容矩阵可只为何发生事务被阻塞。

