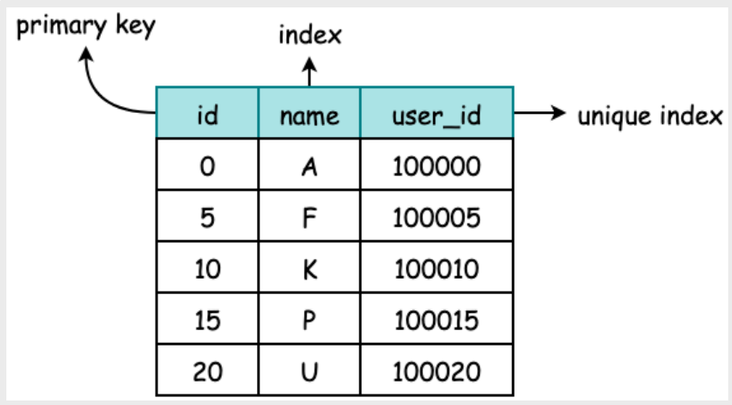
全局锁、表级锁以及行锁（下）

MVCC 与 Lock

- MVCC
  - 多版本并发控制几乎是所有的 RDBMS 的标配，对于提高数据库并发读取的效率有着里程碑式的作用
  - 与 Multi-Version Concurrency Control 对应的是加锁读，即 Lock-Based Concurrency Control，InnoDB 对这两种并发读取方式均有支持
- Lock
  - **innoDB 中采用两阶段加锁，Two-Phase Locking，也就是 2PL。2PL 保证在同一个事务中，加锁和解锁这两个动作绝对不会相交，不同的 SQL 语句在事务进行的不同阶段加锁，而后在事务结束后统一释放锁。也就是说，在事务进行的过程中，绝对不会出现锁释放的情况**
- Next-Key Lock
  - Next-Key Lock 是 Gap Lock 和行级锁的组合，用于在 RR 隔离级别下解决幻读问题。本篇整理的目的就在于详细的阐述 Next-Key Lock 的加锁规则
- 测试数据与环境



  - MySQL Version 8.0.23，RR 隔离级别

```
CREATE TABLE `user` (
  `id` int NOT NULL,
  `name` varchar(16) DEFAULT NULL,
  `user_id` varchar(6) DEFAULT NULL,
  PRIMARY KEY (`id`),
  UNIQUE KEY `user_id` (`user_id`),
  KEY `name` (`name`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;
```

experiment

主键

```
+----+------+----------+
| id | name | user_id |
+----+------+----------+
| 5 | F    | 100005 |
+----+------+----------+
```

```
mysql> begin;
mysql> select * from user where id = 5 for update;
mysql> select * from performance_schema.data_locks\G;
```

```
*************** 1. row ***************
    LOCK_TYPE: TABLE
    LOCK_MODE: IX

***************2. row ***************
    LOCK_TYPE: RECORD
    LOCK_MODE: X,REC_NOT_GAP
LOCK_STATUS: GRANTED
    LOCK_DATA: 5

2 rows in set (0.00 sec)
```

当我们使用主键进行一致性锁定读时，并没有间隙锁的添加，同时也仅仅只是锁住了 id = 5 的这一行数据，也就是说，此时仅在聚簇索引中添加了锁

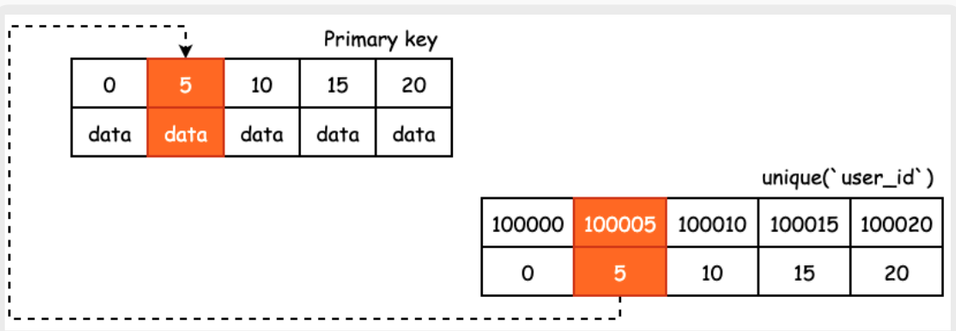
唯一索引

```
+----+------+----------+
| id | name | user_id |
+----+------+----------+
| 5 | F    | 100005 |
+----+------+----------+
```

```
mysql> begin;
mysql> select * from user where user_id = "100005" for update;
mysql> select * from performance_schema.data_locks\G;
```

```
*************** 1. row ***************
    LOCK_TYPE: TABLE
    LOCK_MODE: IX

***************2. row ***************
 INDEX_NAME: user_id
    LOCK_TYPE: RECORD
    LOCK_MODE: X,REC_NOT_GAP
LOCK_STATUS: GRANTED
    LOCK_DATA: '100005', 5

***************3. row ***************
 INDEX_NAME: PRIMARY
    LOCK_TYPE: RECORD
    LOCK_MODE: X,REC_NOT_GAP
LOCK_STATUS: GRANTED
    LOCK_DATA: 5

3 rows in set (0.00 sec)
```

当我们使用唯一索引进行一致性锁定读时，同样也没有间隙锁的添加，但是锁住的范围有所增加，不仅仅是锁住了聚簇索引，唯一索引同样被添加了锁



将两棵 B+Tree 简化一下，将会得到这样的锁定结果

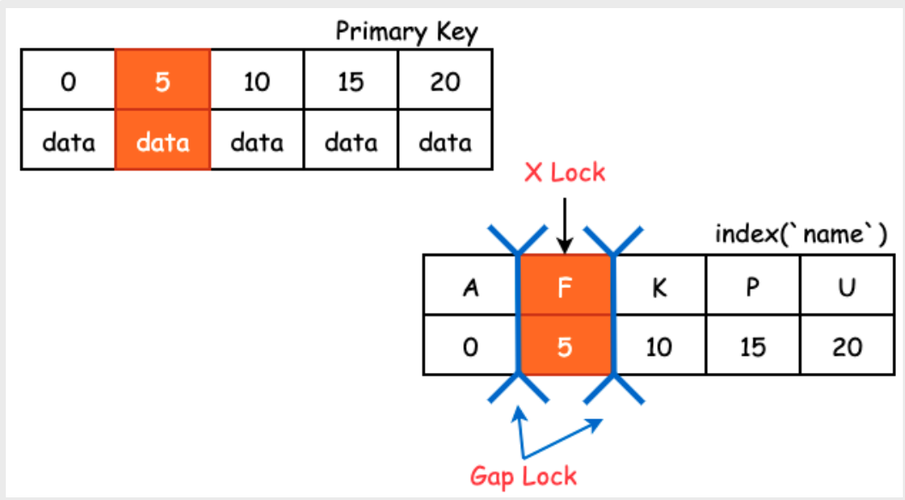
非唯一索引

```
+----+------+----------+
| id | name | user_id |
+----+------+----------+
| 5 | F    | 100005 |
+----+------+----------+
```

```
mysql> begin;
mysql> select * from user where name = "F" for update;
mysql> select * from performance_schema.data_locks\G;
```

```
*************** 1. row ***************
    LOCK_TYPE: TABLE
    LOCK_MODE: IX

***************2. row ***************
 INDEX_NAME: name
    LOCK_TYPE: RECORD
    LOCK_MODE: X
LOCK_STATUS: GRANTED
    LOCK_DATA: 'F', 5

***************3. row ***************
 INDEX_NAME: PRIMARY
    LOCK_TYPE: RECORD
    LOCK_MODE: X,REC_NOT_GAP
LOCK_STATUS: GRANTED
    LOCK_DATA: 5

***************4. row ***************
 INDEX_NAME: name
    LOCK_TYPE: RECORD
    LOCK_MODE: X,GAP
LOCK_STATUS: GRANTED
    LOCK_DATA: 'K', 10

4 rows in set (0.00 sec)
```

当我们使用唯一索引进行一致性锁定读时，一共会有 3 个 X Lock 的添加，除了聚簇索引和二级索引的行级排他锁以外，还会在二级索引中额外添加间隙锁



此时锁定关系如左图所示，那么另一个事务将无法插入被 Gap Lock 所锁住的数据，也就是 name 在 [A, F]，[F, K) 区间中的数据均无法插入