主键选择

自增主键

在 MySQL 中，自增主键是使用的最为频繁主键，我们可以使用 INT 或者是 BIGINT 来作为自增主键

但是，由于 INT 仅占用 4 字节，并且最高位为符号位，因此只能表示 0~2147483648，也就是上限为 21 亿左右。那么如果我们存储的数据可能会超过该值的话，请使用 8 字节的 BIGINT

❶ 自增回溯问题

自增回溯问题出现在 MySQL 8.0 版本之前，出现原因是 MySQL 并不会将每张表的自增值进行持久化，那么一旦 MySQL 发生了异常重启，就可能会出现自增回溯问题

```
+----+------+---------+
| id | name | user_id |
+----+------+---------+
| 1 | Auz  | 100001  |
| 2 | Buz  | 100002  |
| 3 | Cuz  | 100003  |
+----+------+---------+
```

```
mysql> select  * from user;
mysql> delete from user where id = 3;
mysql> show create table user \G;
```

```
****************** 1. row **************
       Table: user
Create Table: CREATE TABLE `user` (
  ......
) ENGINE=InnoDB AUTO_INCREMENT=4
```

当我们删除掉表中的最后一行数据时，下一个自增依然为 4（AUTO_INCREMENT=4），结果正确无误。但是，一旦我们在此时重启了 MySQL 的话，这张表的主键将会发生回溯，也就是 user 表的自增起始值将重新变为 3

```
smartkeyerror@Zero:~$ sudo service mysql restart
```

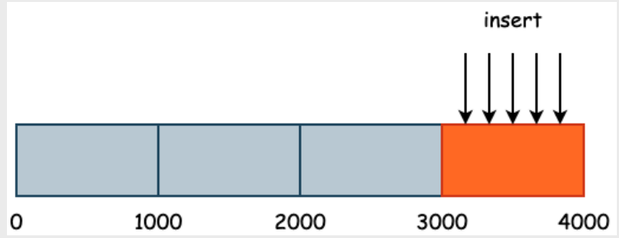
```
mysql> show create table user \G;
```

```
****************** 1. row **************
       Table: user
Create Table: CREATE TABLE `user` (
  ......
) ENGINE=InnoDB AUTO_INCREMENT=3
```

在 MySQL 8.0 之前主键回溯问题并没有得到解决，所以，如果使用自增 ID 最为主键的话，要么升级到 8.0 版本，要么不适用主键 ID 进行任何的业务开展

❷ 基于范围的分区拓展问题

如果我们需要使用自增主键作为分区的 key 时，那么此时我们将不能够使用基于范围的分区方式，因为这会产生热点问题



如左图所示，如果我们的主键是严格自增的，并且使用了基于范围的分区方式，那么数据写入将会按照顺序的方式进行写入，并没有利用到多个分区的优势

并且，在查询数据时，由于最后一个分区总是保存最近插入的数据，那么其查询压力也会大于其它的几个分区

插入、查询都会有热点问题

也就是说，当我们选择了基于范围的分区方式时，仅仅一个主键是不够的，我们不能够将主键作为分区 key 的高位，此时我们需要在主键的前面儿再加点儿东西，使其基于范围分区时能够尽量均匀，并且不会破坏数据插入的有序性

❸ 全局唯一性问题

自增主键只能够保证在当前实例中唯一，但是不能保证在全局范围内唯一。那么此时分区再平衡就很难进行，因为可能会存在主键的冲突

UUID 主键

自增主键除了上面所述的自增回溯问题、基于范围的分区拓展问题以及全局唯一性问题以外，还可能存在安全隐患。若服务端将主键 ID 作为字段返回，那么攻击方将能够轻易的进行数据量的预估，以及进行其它攻击动作

UUID 表示 Universally Unique Identifier，在设计时就是全局唯一的，解决了自增主键无法全局唯一的问题

对于 UUID 版本 1，其组成主要由时间、版本号以及 MAC 地址所组成，并且需要特别注意的是，UUID1 的最高位其实是时间的低位

也就是说，UUID 的高位会随着时间的变化而循环出现，并发单调递增。非单调递增的主键的插入数据时会大致大量的随机 I/O，从而产生性能瓶颈，这也是为什么很少有人直接使用 UUID 作为主键的原因

为了解决这个问题，MySQL 8.0 引入了 UUID_TO_BIN() 方法

1. 将时间高位放在了最前面，解决了插入乱序的问题
2. 去除了无用的 '-' 并且使用二进制的方式对其进行存储，存储空间由 36 字节降低为 16 字节

```
mysql> SELECT UUID_TO_BIN('a6eee2ac-beca-11eb-9413-1c697a6ba8be',TRUE) as UUID_BIN;
+------------------------------------------------+
|                   UUID_BIN                     |
+------------------------------------------------+
| 0x11EBBECAA6EEE2AC94131C697A6BA8BE |
+------------------------------------------------+
1 row in set (0.00 sec)
```

MySQL 8.0 版本的 UUID_TO_BIN() 虽然解决了全局唯一性，但是由于它将时间高位放在了前面，那么 ID 依然会随着时间的增长而增长，在基于范围的分区方式下，依然会有热点问题

分区热点解决

不管是 MySQL 8.0 基于 UUID1 的变种，还是自增主键，它们在使用基于范围的分区时都会存在热点数据问题

此时我们对键进行哈希处理，将得到的哈希值映射到 0~2^32 - 1 这一范围区间内，也就是 MD5(x) % (2^32 - 1)，然后对散列的结果再进行基于范围的分区。因为散列以后我们相当于得到了一个"随机值"，那么使用"随机值"的范围进行分区便可以较大程度上解决某一个分区的热点写入和查询问题