

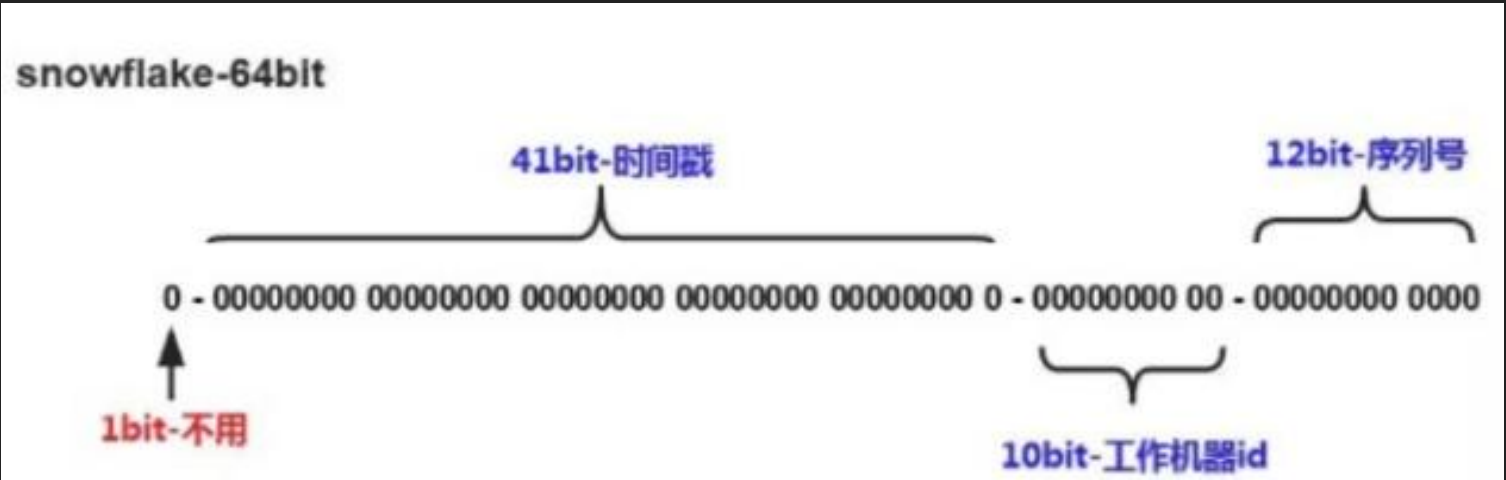
内容大纲

- ▶ 行键设计
- ▶ 列族设计
- ▶ 表设计

行键设计

➤ 唯一原则：行键对应关系型数据库的主键，系统设计之初必须考虑有足够的唯一行键去支持业务的数据量

方案	优点	缺点
自增ID	1、简单易理解、方便查询	1、存在热点问题，新数据会写入同一个Region 2、生成ID的机器存在单点与并发性能问题
雪花算法	1、时间有序性，方便按时间区间扫描 2、高并发，生成ID机器不存在单点问题	1、行键较长，增加存储与处理开销 2、以时间戳作为行键前缀，可能存在热点问题 3、算法相对复杂，生成的ID不利于阅读
UUID	1、高并发，生成ID机器不存在单点问题 2、生产ID分散，不存在热点问题	1、难以排序，只能随机存取，范围查询无法支持 2、不利于阅读，行键无任何业务信息



行键设计

- 长度原则：长度适中，一般从几十到一百字节，建议使用定长，方便从行键提取所需数据，而无须查询出数据内容以节省网络开销

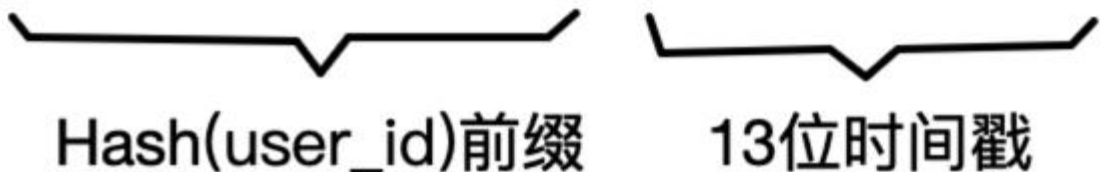


行键设计

- 散列原则：数据是按照行键的字典顺序存储的。如果行键的生成方式使得新的数据都被写入到同一个Region，那么就会导致写入压力不均，出现热点区间问题。为了避免这个问题，行键的设计应该尽量使得数据分散到不同的Region

方法一：hash前缀

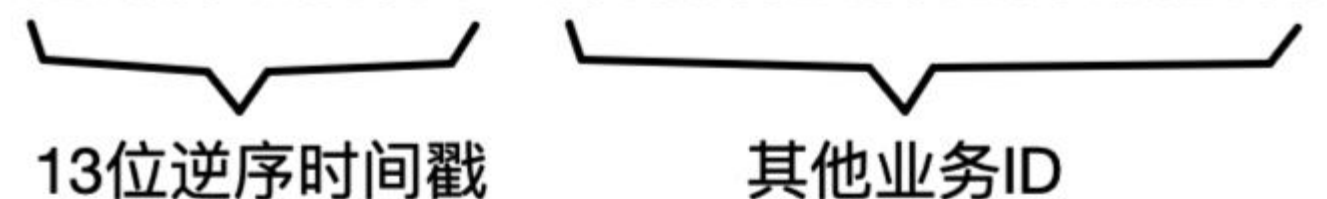
e10adc3949ba59ab 1701314663123



Hash(user_id)前缀 13位时间戳

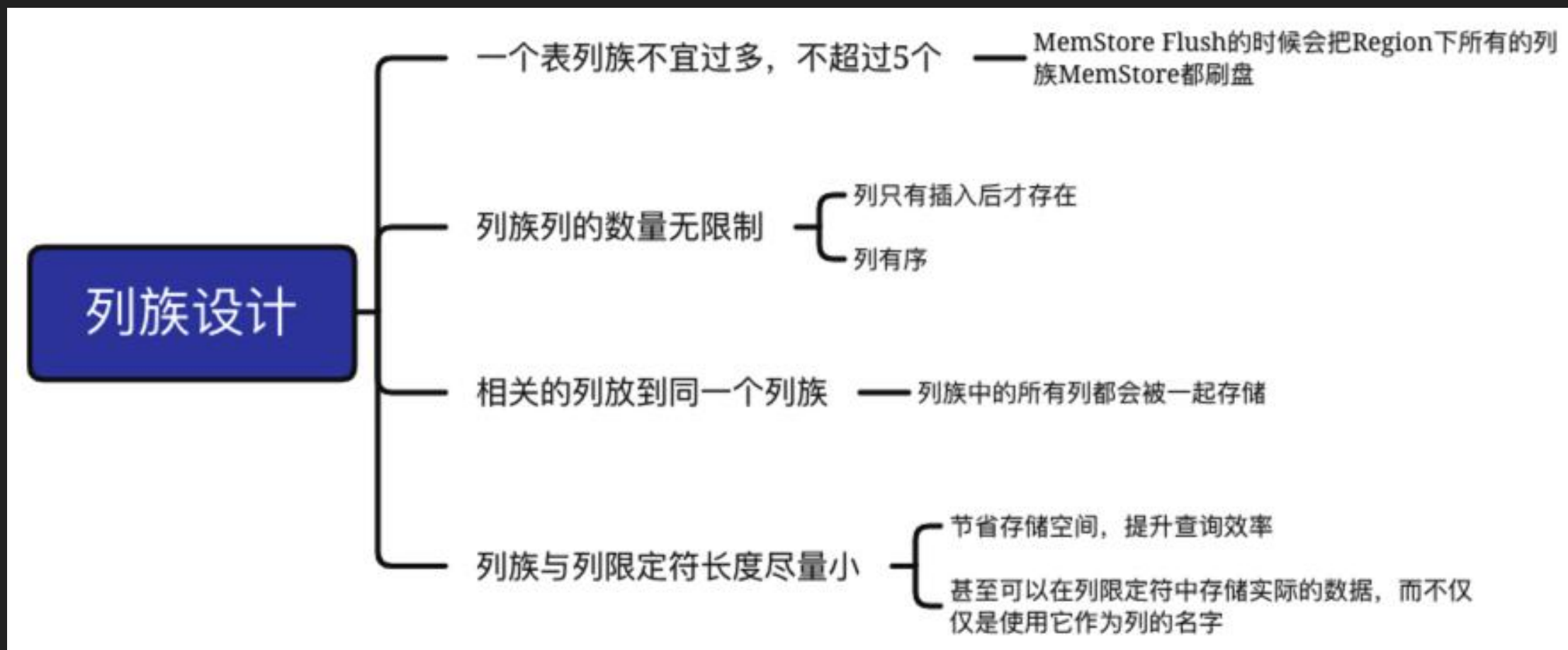
方法二：逆序时间戳

3213664131071 0015963245872354662485



13位逆序时间戳 其他业务ID

列族设计



表设计

模式	描述	优点	缺点	适用场景
高表	数据每行包含的列比较少而行比较多的表	单行查询性能好	表行数多导致需要查询多行数据 行数多可能导致行键设计复杂	适用于需要频繁更新和删除数据的场景
宽表	数据每行包含的列比较多，显得比较胖	所有的列数据都存储在同一行中，可以通过一次查询操作获取。	列的数据分布不均，可能会导致数据热点问题	适合报表分析系统，适合复杂查询与报表生成

表设计

➤ 场景举例

- 云服务-联系人：可以宽表存储，通过`userId`查询一行即可拿到用户所有的联系人
- 用户画像：可以用宽表存储，例如电商网站的用户画像数据，每一行可以是一个用户，列可以是用户的各种属性，如年龄、性别、购买历史等。
- 行为数据：视场景选用高表或者宽表，
 - 高表：一行数据即一次行为，列可以是行为类型、时间、结果等信息。
 - 宽表：一行即一个用户的所有行为，每一次行为可以是1列，列限定符为行为时间戳，内容为行为详情