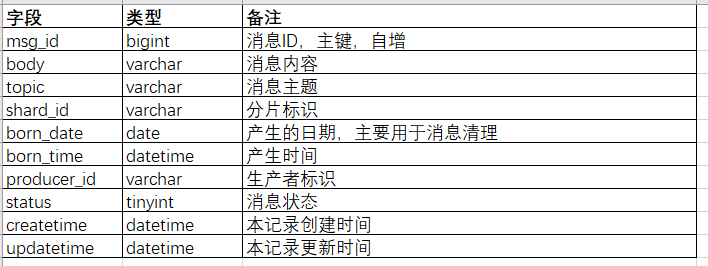
**消息队列存储结构分析**

**message 表**

首先，需要一张表去存储消息体，我们先暂定为 message 表。有几个要点：

* 由于系统架构的设计上数据是进行分组，所以表中需要一个代表分组的字段：shard\_id。每一组数据分组集群，其 shard\_id 是固定的。这个类似于 kafka 的 partition 分区设计，不同的 consumer 可以拉取不同的 shard，达到加速消费的目的；
* 此处设计每个 topic 用一张单独的表去存储，即 **message\_{topic}**，原因是 1.MySQL 理论上支持的表上亿，能满足 topic 数量要求 2.用单张表存可以隔离各 topic 的影响（物理上的逻辑上的）3.使用起来比较简单，因为单表中只可能是该 topic 的消息，id 也可以使用自增的。

所以最终消息表的设计如下：

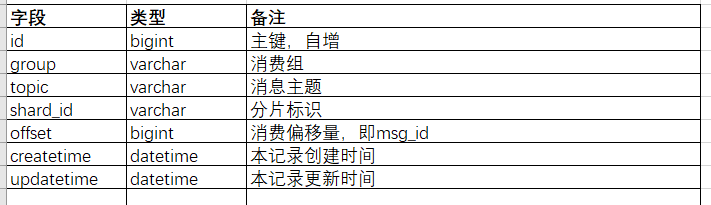


**consume\_log 表**

光有消息表还不够，还需要一张表记录消费状态。也有几个要点：

* 我们支持多个消费者组，每个消费者组共享一条消费进度记录，所以表中需要有 group 字段来区分是哪个消费者组的消费记录；
* 消费偏移量是最重要的，记录了消费到哪条数据，用 offset 来标识，即 message 表中的 msg\_id，因为 msg\_id 在一个数据分组，一个 topic 下是严格递增的；

所以最终消息消费记录表设计如下：



如果不考虑太复杂的设计，核心主要就这两个表。

**索引分析**

我们先从几个高频场景来分析：

**发送消息**

1. 客户端完成分片路由，分配到某个 shard，发送消息数据
2. 消息队列服务器根据 topic 找到对应的消息表 message\_{topic}，然后插入数据，msg\_id 自增

**消费消息**

1. 客户端完成分片的路由，向对应服务器拉取消息，带上参数 topic，group
2. 查询 consumer\_log 表，根据 topic，group 通过索引找到对应的数据，取出 offset
3. 根据 offset 查询对应的 message 表，通过主键 id，找出下个 offset 的记录，取出并准备返回
4. 更新 consumer\_log 的 offset 信息
5. 返回数据

以上 2，3，4 在一个事务中执行。

从 2 可以看出，message\_log 需要加个 topic+group 组合索引，命名为 idx\_topic\_group

从 3 可以看出，message 表需要 msg\_id 的索引，但是其本身已经是主键，无需额外增加

**消息过期**

1.定时任务定期启动（如 1 天），计算需要清理的日期值，扫描 message 表，找到需要过期记录，批量更新状态为删除（或者直接物理删除）

此处可以看出为了方便快速扫描到要过期的记录，message 的 born\_date 需要加个索引，命名为 idx\_born\_date

**索引总结**

所以根据分析，需要如下索引：

1. message\_log 表增加 **idx\_topic\_group**，是 topic+group 的组合索引
2. message 表增加 **idx\_born\_date**，是 born\_date 的索引