我们平时开发中不可避免的就是要存储时间，比如我们要记录操作表中这条记录的时间、记录转账的交易时间、记录出发时间等等。你会发现这个时间这个东西与我们开发的联系还是非常紧密的，用的好与不好会给我们的业务甚至功能带来很大的影响。所以，我们有必要重新出发，好好认识一下这个东西。

这是一篇短小精悍的文章，仔细阅读一定能学到不少东西！

1.切记不要用字符串存储日期

我记得我在大学的时候就这样干过，而且现在很多对数据库不太了解的新手也会这样干，可见，这种存储日期的方式的优点还是有的，就是简单直白，容易上手。

但是，这是不正确的做法，主要会有下面两个问题：

1. 字符串占用的空间更大！
2. 字符串存储的日期比较效率比较低（逐个字符进行比对），无法用日期相关的 API 进行计算和比较。

2.Datetime 和 Timestamp 之间抉择

Datetime 和 Timestamp 是 MySQL 提供的两种比较相似的保存时间的数据类型。他们两者究竟该如何选择呢？

通常我们都会首选 Timestamp。 下面说一下为什么这样做!

2.1 DateTime 类型没有时区信息的

DateTime 类型是没有时区信息的（时区无关） ，DateTime 类型保存的时间都是当前会话所设置的时区对应的时间。这样就会有什么问题呢？当你的时区更换之后，比如你的服务器更换地址或者更换客户端连接时区设置的话，就会导致你从数据库中读出的时间错误。不要小看这个问题，很多系统就是因为这个问题闹出了很多笑话。

Timestamp 和时区有关。Timestamp 类型字段的值会随着服务器时区的变化而变化，自动换算成相应的时间，说简单点就是在不同时区，查询到同一个条记录此字段的值会不一样。

下面实际演示一下！

建表 SQL 语句：

```
CREATE TABLE `time_zone_test` (
  `id` bigint(20) NOT NULL AUTO_INCREMENT,
  `date_time` datetime DEFAULT NULL,
  `time_stamp` timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE
CURRENT_TIMESTAMP,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

插入数据：

```
INSERT INTO time_zone_test(date_time,time_stamp) VALUES(NOW(),NOW());
```

查看数据：

```
select date_time,time_stamp from time_zone_test;
```

结果：

```
+---------------------+---------------------+
| date_time           | time_stamp          |
+---------------------+---------------------+
| 2020-01-11 09:53:32 | 2020-01-11 09:53:32 |
+---------------------+---------------------+
```

现在我们运行

修改当前会话的时区:

```
set time_zone='+8:00';
```

再次查看数据：

```
+---------------------+---------------------+
| date_time           | time_stamp          |
+---------------------+---------------------+
| 2020-01-11 09:53:32 | 2020-01-11 17:53:32 |
+---------------------+---------------------+
```

扩展：一些关于 MySQL 时区设置的一个常用 sql 命令

```
# 查看当前会话时区
SELECT @@session.time_zone;
# 设置当前会话时区
SET time_zone = 'Europe/Helsinki';
SET time_zone = "+00:00";
# 数据库全局时区设置
SELECT @@global.time_zone;
# 设置全局时区
SET GLOBAL time_zone = '+8:00';
SET GLOBAL time_zone = 'Europe/Helsinki';
```

2.2 DateTime 类型耗费空间更大

Timestamp 只需要使用 4 个字节的存储空间，但是 DateTime 需要耗费 8 个字节的存储空间。但是，这样同样造成了一个问题，Timestamp 表示的时间范围更小。

- DateTime ： 1000-01-01 00:00:00 ~ 9999-12-31 23:59:59
- Timestamp： 1970-01-01 00:00:01 ~ 2037-12-31 23:59:59

> Timestamp 在不同版本的 MySQL 中有细微差别。

3 再看 MySQL 日期类型存储空间

下图是 MySQL 5.6 版本中日期类型所占的存储空间：

Date and Time Type Storage Requirements

For `TIME`, `DATETIME`, and `TIMESTAMP` columns, the storage required for tables created before MySQL 5.6.4 differs from tables created from 5.6.4 on. This is due to a change in 5.6.4 that permits these types to have a fractional part, which requires from 0 to 3 bytes.

Data Type	Storage Required Before MySQL 5.6.4	Storage Required as of MySQL 5.6.4
`YEAR`	1 byte	1 byte
`DATE`	3 bytes	3 bytes
`TIME`	3 bytes	3 bytes + fractional seconds storage
`DATETIME`	8 bytes	5 bytes + fractional seconds storage
`TIMESTAMP`	4 bytes	4 bytes + fractional seconds storage

As of MySQL 5.6.4, storage for `YEAR` and `DATE` remains unchanged. However, `TIME`, `DATETIME`, and `TIMESTAMP` are represented differently. `DATETIME` is packed more efficiently, requiring 5 rather than 8 bytes for the nonfractional part, and all three parts have a fractional part that requires from 0 to 3 bytes, depending on the fractional seconds precision of stored values.

Fractional Seconds Precision	Storage Required
0	0 bytes
1, 2	1 byte
3, 4	2 bytes
5, 6	3 bytes

For example, `TIME(0)`, `TIME(2)`, `TIME(4)`, and `TIME(6)` use 3, 4, 5, and 6 bytes, respectively. `TIME` and `TIME(0)` are equivalent and require the same storage.

For details about internal representation of temporal values, see MySQL Internals: Important Algorithms and Structures.

可以看出 5.6.4 之后的 MySQL 多出了一个需要 0 ~ 3 字节的小数位。Datatime 和 Timestamp 会有几种不同的存储空间占用。

为了方便，本文我们还是默认 Timestamp 只需要使用 4 个字节的存储空间，但是 DateTime 需要耗费 8 个字节的存储空间。

4.数值型时间戳是更好的选择吗?

很多时候，我们也会使用 int 或者 bigint 类型的数值也就是时间戳来表示时间。

这种存储方式的具有 Timestamp 类型的所具有一些优点，并且使用它的进行日期排序以及对比等操作的效率会更高，跨系统也很方便，毕竟只是存放的数值。缺点也很明显，就是数据的可读性太差了，你无法直观的看到具体时间。

时间戳的定义如下：

> 时间戳的定义是从一个基准时间开始算起，这个基准时间是「1970-1-1 00:00:00 +0:00」，从这个时间开始，用整数表示，以秒计时，随着时间的流逝这个时间整数不断增加。这样一来，我只需要一个数值，就可以完美地表示时间了，而且这个数值是一个绝对数值，即无论的身处地球的任何角落，这个表示时间的时间戳，都是一样的，生成的数值都是一样的，并且没有时区的概念，所以在系统的中时间的传输中，都不需要进行额外的转换了，只有在显示给用户的时候，才转换为字符串格式的本地时间。

数据库中实际操作：

```
mysql> select UNIX_TIMESTAMP('2020-01-11 09:53:32');
+---------------------------------------+
| UNIX_TIMESTAMP('2020-01-11 09:53:32') |
+---------------------------------------+
|                            1578707612 |
+---------------------------------------+
1 row in set (0.00 sec)

mysql> select FROM_UNIXTIME(1578707612);
+----------------------------+
| FROM_UNIXTIME(1578707612) |
+----------------------------+
| 2020-01-11 09:53:32        |
+----------------------------+
1 row in set (0.01 sec)
```

5.总结

MySQL 中时间到底怎么存储才好？Datetime?Timestamp? 数值保存的时间戳?

好像并没有一个银弹，很多程序员会觉得数值型时间戳是真的好，效率又高还各种兼容，但是很多人又觉得它表现的不够直观。这里插一嘴，《高性能 MySQL 》这本神书的作者就是推荐 Timestamp，原因是数值表示时间不够直观。下面是原文：

除了特殊行为之外，通常也应该尽量使用 TIMESTAMP，因为它比 DATETIME 空间效率更高。有时候人们会将 Unix 时间截存储为整数值，但这不会带来任何收益。用整数保存时间截的格式通常不方便处理，所以我们不推荐这样做。

如果需要存储比秒更小粒度的日期和时间值怎么办？ MySQL 目前没有提供合适的数据类型，但是可以使用自己的存储格式：可以使用 BIGINT 类型存储微秒级别的时间截，或者使用 DOUBLE 存储秒之后的小数部分。这两种方式都可以，或者也可以使用 MariaDB 替代 MySQL。

每种方式都有各自的优势，根据实际场景才是王道。下面再对这三种方式做一个简单的对比，以供大家实际开发中选择正确的存放时间的数据类型：

日期类型	存储空间	日期格式	日期范围	是否存在时区问题
Datetime	8字节	YYYY-MM-DD HH:MM:SS	1000-01-01 00:00:00 ~ 9999-12-31 23:59:59	是
Timestamp	4 字节	YYYY-MM-DD HH:MM:SS	1970-01-01 00:00:00 ~ 2037-12-31 23:59:59	否
时间戳	4字节	全数字如1578707612	1970-01-01 00:00:01 之后的时间	否

作者：SnailClimb
公众号&Github：JavaGuide

如果还有什么问题欢迎给我留言！如果文章有什么问题的话，也劳烦指出，Guide 哥感激不尽！

后面的文章我会介绍：

- [ ] Java8 对日期的支持以及为啥不能用 SimpleDateFormat。
- [ ] SpringBoot 中如何实际使用(JPA 为例)