在执行MySQL时发现了一个有意思的错误：

假设有表x, 表y 结构如下：

```
CREATE TABLE `x` (
`id` int(10) unsigned NOT NULL AUTO_INCREMENT,
`name` varchar(50) COLLATE utf8mb4_unicode_ci NOT NULL DEFAULT '' COMMENT 'name',
PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_unicode_ci COMMENT='x';

CREATE TABLE `y` (
`id` int(10) unsigned NOT NULL AUTO_INCREMENT,
`name` varchar(50) CHARACTER SET utf8mb4 DEFAULT NULL COMMENT 'name',
PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_unicode_ci COMMENT='y';
```

select * from x left join y on x.name=y.name;

执行一条mysql left join y表on x.name=y.name后客户端得到错误：

```
[HY000][1267] Illegal mix of collations (utf8mb4_general_ci,IMPLICIT) and (utf8mb4_unicode_ci,IMPLICIT) for operation '='
```

可以看到貌似提示编码错误！

```
show variables  like '%char%';

Variable_name,Value
character_set_client,utf8mb4
character_set_connection,utf8mb4
character_set_database,utf8mb4
character_set_filesystem,binary
character_set_results,utf8mb4
character_set_server,utf8
character_set_system,utf8
character_sets_dir,/usr/local/mysql56/share/charsets/
```

我们统一使用占用4字节的utf8mb4编码。

我们知道MySQL数据存储底层其实btree结构，在数据的查询，由客户端发起请求，server接受请求，直至在btree中检索数据其实整个过程涉及到好几处编码转换，数据库连接，查找处理，结果返回都涉及到好几种编码转换。

如果定义编码不一致那么就可能得到不正确的结果返回，比如使用utf8mb4存储数据，使用utf8的客户端去连接并接受返回，那么最终的结果可能会丢掉一些字节。

如果是熟悉安全的同学，可能知道这些编码转换问题在有些时候能够吞掉一些字节，制造出一些奇妙的注入。

这里说一下字符编码，MySQL默认有四种级别的字符集：

- 服务器级别
- 数据库级别
- 表级别
- 列级别

默认utf8编码其实是utf8mb3,这个是正宗utf8编码的的一个特殊形式，占用3个字节，而utfbmb4占用4个字节。gb2312占用2个字节，gbk占用2个字节（在gbk上增加了点内容）。ascii只占用一个字节。

MySQL在数据查询时，客户端会先申明自己请求时的字符集，MySQL本身在数据存储时由于编码的不同，那么数据的排序其实也是不同的，因为建立索引的过程其实就是数据排序的过程，这里的排序是指为了维护btree结构在硬盘上的物理排序，不是数据表结果可视化后给你看到的排序。

很明显，我们之所以这条简单的SQL都执行不成功就是因为x,y表的collate不同：

```
`name` varchar(50) COLLATE utf8mb4_unicode_ci NOT NULL DEFAULT ''
```

```
`name` varchar(50) CHARACTER SET utf8mb4 DEFAULT NULL
```

collate一个是utf8mb4_unicode_ci，一个是utf8mb4

其实你只要改成同样的collate就能work了。另外，题外话，MySQL底层其实是非常复杂的，在千万乃至亿级数据时，内部维护的btree的高度也就3，4层，可想而知！