MySQL 选择合适的数据类型

1. CHAR与VARCHAR

CHAR 和 VARCHAR 类型类似，都用来存储字符串，但它们保存和检索的方式不同。CHAR 属于固定长度的字符类型，而 VARCHAR 属于可变长度的字符类型。

CHAR 和 VARCHAR 的对比

----------------------------------------------------------------------------------------------------------------------------------

值 | CHAR（4） | 存储需求 | VARCHAR（4） | 存储需求

----------------------------------------------------------------------------------------------------------------------------------

'' | ' ' | 4 个字节 | ‘’ | 1 个字节

'ab' | 'ab ' | 4 个字节 | ‘ab' | 3 个字节

'abcd' | 'abcd' | 4 个字节 | 'abcd' | 5 个字节

'abcdefg' | 'abcd' | 4 个字节 | 'abcd' | 5 个字节

----------------------------------------------------------------------------------------------------------------------------------

由于 CHAR 是固定长度的，所以它的处理速度比 VARCHAR 快得多，但是其缺点是浪费存储空间，程序需要对行尾空格进行处理，所以对于那些长度变化不大且对查询速度有较高要求的数据可以考虑用 CHAR 类型来存储。

另外，在MySQL中，不同的存储引擎对 CHAR 和 VARCHAR 的使用原则有所不同，概括如下：

1）MyISAM 存储引擎：建议使用固定长度的数据列代替可变长度的数据列；

2）MEMORY存储引擎：目前都使用固定长度的数据行存储，因此无论使用 CHAR 或 VARCHAR 列都没有关系。两者都是作为 CHAR 类型处理；

3）InnoDB存储引擎：建议使用 VARCHAR 类型。对于InnoDB数据表，内部的行存储格式没有区分固定长度和可变长度列（所有数据行都使用指向数据列值的头指针），因此在本质上，使用固定长度的 CHAR 列不一定比使用可变长度 VARCHAR 列性能要好。因此，主要的性能因素是数据行使用的存储总量。由于 CHAR 平均占用的空间多于 VARCHAR，因此使用 VARCHAR 来最小化需要处理的数据行的存储总量和磁盘 I/O 是比较好的。

2. TEXT 与 BLOB

一般保存少量字符串的时候，可以选择 CHAR 或者 VARCHAR，而在保存大文本时，通常会选择使用 TEXT 或者 BLOB，二者之间的主要差别是 BLOB 能用来保存二进制数据，比如照片；而 TEXT 只能保存字符数据，比如一篇文章或者日记。

BLOB 与 TEXT 存在的一些常见的问题：

1）BLOB 和 TEXT 值会引起一些性能问题，特别是执行了大量的删除操作时。

删除操作会在数据表中留下很大的“空洞”，以后填入这些“空洞”的记录在插入的性能上会有影响。为了提高性能，建议定期使用 OPTIMIZE TABLE 功能对这类表进行碎片整理，避免因为“空洞”导致性能问题。

2）使用合成的（Synthetic）索引来提高大文本字段（BLOB 或 TEXT）的查询性能。

简单点说，合成索引就是根据大文本字段的内容建立一个散列值，并把这个值存储在单独的数据列中，接下来就可以通过检索散列值找到数据行了。但是，要注意这种技术只能用于精确匹配的查询（散列值对于类似< 或者 >= 等范围搜索操作符是没有用处的）。可以使用MD5() 函数生成散列值，也可以使用SHA1() 或 CRC32()，或者使用自己的应用程序逻辑来计算散列值。请记住数值型散列值可以很高效率地存储。同样，如果散列算法生成的字符串带有尾部空格，就不要把它们存在在 CHAR 或 VARCHAR 列中，它们会受到尾部空格去除的影响。合成的散列索引对于那些BLOB或TEXT数据列特别有用。用散列标识符值查找的速度比搜索BLOB列本身的速度快很多。

3）在不必要的时候避免检索大型的BLOB或TEXT值。

例如，SELECT \* 查询就不是很好的想法，除非能够确定作为约束条件的WHERE子句只会找到所需要的数据行，否则，很可能毫无目的在网络上传输大量的值。

4）把 BLOB 或 TEXT 列分离到单独的表中。

在某些环境中，如果把这些数据列移动到第二张数据表中，可以把原数据表中的数据列转换为固定长度的数据行格式，那么它就是有意义的。这会减少主表中的碎片，可以得到固定长度数据行的性能优势。它还可以使主数据表在运行 SELECT \* 查询的时候不会通过网络传输大量的 BLOB 或 TEXT 值。

3. 浮点数与定点数

浮点数一般用于表示含有小数部分的数值。当一个字段被定义为浮点类型后，如果插入数据的精度超过该列定义的实际精度，则插入值会被四舍五入到实际定义的精度值，然后插入，四舍五入的过程不会报错。在MySQL中，float, double（或real）用来表示浮点数。

定点数实际上是以字符串形式存放的，所以定点数可以更精确地保存数据。如果实际插入的数值精度大于实际定义的精度，则MySQL会进行警告（默认的SQLMode下），但是数据按照实际精度四舍五入后插入；如果SQLMode是在TRADITIONAL（传统模式）下，则系统会直接报错，导致数据无法插入。在MySQL中，decimal（或numberic）用来表示定点数。

下面，以一个例子来看浮点数的精确性问题。

mysql > CREATE TABLE t1 (f float(8, 1));

mysql > INSERT INTO t1 VALUES(1.23456);

mysql > INSERT INTO t1 VALUES(1.25456);

mysql > SELECT \* FROM t1;

从最后一个 mysql 语句运行的结果看，对于第一次插入值1.23456到float(8, 1)时，该值被截断，并保存为1.2；而第二次插入值1.25456到float(8, 1)时，该值进行了四舍五入然后被截断，并保存为1.3，所以在选择浮点型数据保存小数时，要注意四舍五入的问题，并尽量保留足够的小数位，避免存储的数据不准确。

下面的例子，说明了浮点数与定点数的区别。

mysql > CREATE TABLE t2 (c1 float(10, 2), c2 decimal(10, 2));

mysql > INSERT INTO t2 VALUES(131072.32, 131072.32);

mysql > SELECT \* FROM t2;

结果是：131072.31 131072.32

从上面的例子可以看出，c1 列的值由131072.32变成了131072.31，这是上面是数值在使用单精度浮点数表示时，产生了误差。这是浮点数特有的问题，因此在精度要求比较高的应用中（比如货币），要使用定点数而不是浮点数来保存数据。

总结来说，在以后关于浮点数与定点数的应用中，考虑一下几个原则：

1）浮点数存在误差问题；

2）对货币等精度敏感的数据，应该用定点数表示或存储；

3）在编程中，如果用到浮点数，要特别注意误差问题，并尽量避免浮点数比较；

4）要注意浮点数中一些特殊值的处理；

4. 日期类型选择

MySQL中提供的常用日期类型有DATE，TIME，DATETIME，TIMESTAMP。选择日期类型的原则有：

1）根据实际需要选择能够满足应用的最小存储的日期类型。如果应用只需要记录“年份”，那么用1个字节来存储的YEAR类型完全能够满足，而不需要用4个字节来存储的DATE类型。这样不仅仅能节约存储，更能够提高表的操作效率；

2）如果要记录年月日时分秒，并且记录的年份比较久远，那么最好使用DATETIEM，而不要使用TIMESTAMP。因为TIMESTAMP表示的日期范围比DATETIME要短的多；

3）如果记录的日期需要让不同时区的用户使用，那么最好使用TIMESTAMP，因为日期类型中只有它能够和实际时区相对应。

5. 总结

1）对于字符类型，要根据存储引擎来进行相应的选择；

2）对精度要求较高的应用中，建议使用定点数来存储数值，以保证结果的准确性。

3）对含有 TEXT 和 BLOB 字段的表，如果经常做删除和修改记录的操作，则要定时执行 OPTIMIZE TABLE 功能对表进行碎片整理；

4）日期类型要根据实际需要选择能够满足应用的最小存储的日期类型。