

A.5.3. Problems with NULL Values

The concept of the `NULL` value is a common source of confusion for newcomers to SQL, who often think that `NULL` is the same thing as an empty string `' '`. This is not the case. For example, the following statements are completely different:

```
mysql> INSERT INTO my_table (phone) VALUES (NULL);
mysql> INSERT INTO my_table (phone) VALUES ( '');
```

Both statements insert a value into the `phone` column, but the first inserts a `NULL` value and the second inserts an empty string. The meaning of the first can be regarded as "phone number is not known" and the meaning of the second can be regarded as "the person is known to have no phone, and thus no phone number."

To help with `NULL` handling, you can use the `IS NULL` and `IS NOT NULL` operators and the `IFNULL()` function.

In SQL, the `NULL` value is never true in comparison to any other value, even `NULL`. An expression that contains `NULL` always produces a `NULL` value unless otherwise indicated in the documentation for the operators and functions involved in the expression. All columns in the following example return `NULL`:

```
mysql> SELECT NULL, 1+NULL, CONCAT('Invisible',NULL);
```

If you want to search for column values that are `NULL`, you cannot use an `expr = NULL` test. The following statement returns no rows, because `expr = NULL` is never true for any expression:

```
mysql> SELECT * FROM my_table WHERE phone = NULL;
```

To look for `NULL` values, you must use the `IS NULL` test. The following statements show how to find the `NULL` phone number and the empty phone number:

```
mysql> SELECT * FROM my_table WHERE phone IS NULL;
mysql> SELECT * FROM my_table WHERE phone = ' ';
```

You can add an index on a column that can have `NULL` values if you are using MySQL 3.23.2 or newer and are using the `MyISAM`, `InnoDB`, or `BDB` storage engine. As of MySQL 4.0.2, the `MEMORY` storage engine also supports `NULL` values in indexes. Otherwise, you must declare an indexed column `NOT NULL` and you cannot insert `NULL` into the column.

When reading data with `LOAD DATA INFILE`, empty or missing columns are updated with `' '`. If you want a `NULL` value in a column, you should use `\N` in the data file. The literal word "NULL" may also be used under some circumstances. See [Section 13.1.5, "LOAD DATA INFILE Syntax"](#).

When using `DISTINCT`, `GROUP BY`, or `ORDER BY`, all `NULL` values are regarded as equal.

When using `ORDER BY`, `NULL` values are presented first, or last if you specify `DESC` to sort in descending order. Exception: In MySQL 4.0.2 through 4.0.10, `NULL` values sort first regardless of sort order.

Aggregate (summary) functions such as `COUNT()`, `MIN()`, and `SUM()` ignore `NULL` values. The exception to this is `COUNT(*)`, which counts rows and not individual column values. For example, the following statement produces two counts. The first is a count of the number of rows in the table, and the second is a count of the number of non-`NULL` values in the `age` column:

```
mysql> SELECT COUNT(*), COUNT(age) FROM person;
```

For some column types, MySQL handles `NULL` values specially. If you insert `NULL` into a `TIMESTAMP` column, the current date and time is inserted. If you insert `NULL` into an integer column that has the `AUTO_INCREMENT` attribute, the next number in the sequence is inserted.