If you access a column from the table to be updated in an expression, `UPDATE` uses the current value of the column. The second assignment in the following statement sets `col2` to the current (updated) `col1` value, not the original `col1` value. The result is that `col1` and `col2` have the same value. This behavior differs from standard SQL.

```
UPDATE t1 SET col1 = col1 + 1, col2 = col1;
```

1.7.2.3 FOREIGN KEY Constraint Differences

The MySQL implementation of foreign key constraints differs from the SQL standard in the following key respects:

- If there are several rows in the parent table with the same referenced key value, `InnoDB` performs a foreign key check as if the other parent rows with the same key value do not exist. For example, if you define a `RESTRICT` type constraint, and there is a child row with several parent rows, `InnoDB` does not permit the deletion of any of the parent rows.

- If `ON UPDATE CASCADE` or `ON UPDATE SET NULL` recurses to update the *same table* it has previously updated during the same cascade, it acts like `RESTRICT`. This means that you cannot use self-referential `ON UPDATE CASCADE` or `ON UPDATE SET NULL` operations. This is to prevent infinite loops resulting from cascaded updates. A self-referential `ON DELETE SET NULL`, on the other hand, is possible, as is a self-referential `ON DELETE CASCADE`. Cascading operations may not be nested more than 15 levels deep.

- In an SQL statement that inserts, deletes, or updates many rows, foreign key constraints (like unique constraints) are checked row-by-row. When performing foreign key checks, `InnoDB` sets shared row-level locks on child or parent records that it must examine. MySQL checks foreign key constraints immediately; the check is not deferred to transaction commit. According to the SQL standard, the default behavior should be deferred checking. That is, constraints are only checked after the *entire SQL statement* has been processed. This means that it is not possible to delete a row that refers to itself using a foreign key.

- No storage engine, including `InnoDB`, recognizes or enforces the `MATCH` clause used in referential-integrity constraint definitions. Use of an explicit `MATCH` clause does not have the specified effect, and it causes `ON DELETE` and `ON UPDATE` clauses to be ignored. Specifying the `MATCH` should be avoided.

  The `MATCH` clause in the SQL standard controls how `NULL` values in a composite (multiple-column) foreign key are handled when comparing to a primary key in the referenced table. MySQL essentially implements the semantics defined by `MATCH SIMPLE`, which permits a foreign key to be all or partially `NULL`. In that case, a (child table) row containing such a foreign key can be inserted even though it does not match any row in the referenced (parent) table. (It is possible to implement other semantics using triggers.)

- MySQL requires that the referenced columns be indexed for performance reasons. However, MySQL does not enforce a requirement that the referenced columns be `UNIQUE` or be declared `NOT NULL`.

  A `FOREIGN KEY` constraint that references a non-`UNIQUE` key is not standard SQL but rather an `InnoDB` extension. The `NDB` storage engine, on the other hand, requires an explicit unique key (or primary key) on any column referenced as a foreign key.

  The handling of foreign key references to nonunique keys or keys that contain `NULL` values is not well defined for operations such as `UPDATE` or `DELETE CASCADE`. You are advised to use foreign keys that reference only `UNIQUE` (including `PRIMARY`) and `NOT NULL` keys.

- MySQL parses but ignores “inline `REFERENCES` specifications” (as defined in the SQL standard) where the references are defined as part of the column specification. MySQL accepts `REFERENCES` clauses