Time and MySQL

Who knows where the time goes? [1]

A basic feature of the relational database is row uniqueness as guaranteed by a table's primary key. Without it, we can't tell one row from another, and we can't relate rows in one table to rows in other tables.

What happens to row uniqueness, and to relational properties that depend on it, when a set of tables must model variation over time? On the face of it, you might expect this, at worst, to add a few innocent complexities—a datetime column or two per table, perhaps, and an extra line or two in query clauses. Then ask yourself how to write a constraint that permits multiple rows to have identical non-datetime values at different instants, and to have overlapping valid periods, *but never the same non-temporal data at the same instant*. Good luck working that out on the back of an envelope.

This chapter is about how to model such time variation in a MySQL database. We begin as if the reader were a client, with the bad news. When it comes to modelling time in a database, complexities multiply quickly. Because SQL has no universal quantifier, nested negatives abound. There aren't many brilliant shortcuts to be had. Mostly, working out these constraints amounts to step-by-step slogging.

Now the good news: there are simple, basic organising concepts.

Basic Concepts

To specify validity in time is to build *temporal relational database architecture*. If you have tried this in SQL, you appreciate how difficult it is. In MySQL, the unavailability of CHECK CONSTRAINTs and DEFERRED CONSTRAINTs, and some limits on triggers, make it even harder.

To develop temporal database architecture we analyse time-sensitive data on three basic temporal dimensions: *domain type*, *value reference*, and *validity type*:

A. **Temporal domain types**: The three kinds are:
   - *instant*, a temporal atom (e.g., a microsecond, a second, a day),
   - *interval*, a duration between two instants,
   - *period*, an interval anchored to a particular instant.

   An interval or period is a sequence of temporal atoms. It may be:
   - *closed-closed*: includes both start date and end date,
   - *closed-open*: includes the start date, excludes the end date,
   - *open-closed*: excludes the start date and includes the end date,
   - *open-open*: excludes both start date and end date.

   Inclusive/exclusive would be more intuitive than *closed/open*, but we follow the standard nomenclature. A common business default is *closed-open*: if you book a hotel room for 22-24 May, the hotel will expect you to arrive in the afternoon of 22 May and leave in the morning of 24 May, having stayed the two days of the *closed:closed* period 22-23 May, or the *closed:open* period 22-24 May. Obviously a temporal database needs one consistent convention. We will stick to *closed-open* periods.

B. **Temporal value references**: The three kinds are:
   - *user-defined* or arbitrary, orthogonal to the validity of other columns,
   - *valid time*, marking when a stored fact was so, ie *historical facts*;
   - *transaction time*, marking when a fact was stored; tracking it is often described as *point-in-time architecture* (PITA).

C. **Temporal validity**: The three kinds are:
   - *current*: now,
   - *sequenced*: at each instant in the relevant datetime range,
   - *non-sequenced*: time-independent.

Thus queries on temporal tables
- may be current, sequenced or non-sequenced,
- may reference user-defined time, valid time, and/or transaction time, and
- may return snapshots or period information.

MySQL limitations

CHECK CONSTRAINT

A `CHECK CONSTRAINT` is often of the form

```
CHECK( [NOT] EXISTS( select_expression ))
```

MySQL implements foreign key constraints in INNODB tables, but does not yet implement `CHECK CONSTRAINT`. Until it does, such constraints must be enforced by other means. That has onerous consequences for time-valid tables. Some time-valid constraints can be enforced in triggers, but most of the temporal constraints we will consider cannot. Until MySQL implements CHECK CONSTRAINT, they must be enforced in application code. That is a heavy penalty.

Deferred constraints

MySQL does not yet implement deferred constraints, either. Furthermore, constraints are applied row-wise rather than at `COMMIT` time. This raises a problem for many complex constraints, even for some simple ones. For example to delete a MySQL row which refers to itself via a foreign key, you must temporarily `SET foreign_key_checks = 0`. A transaction fulfilling a complex constraint must leave the database in a consistent state. But there is nothing in relational database theory to suggest that a database should be in a consistent state after each statement within a transaction.

Triggers

MySQL 5 triggers cannot issue UPDATE statements on the trigger table, and cannot raise errors. These limitations create difficulties for implementing transaction validity in MySQL, but the difficulties can be overcome.

TOC ***Previous*** ***Next***