Prof. Dr. Florian Heinz
florian.heinz@oth-regensburg.de

Modern Database Concepts

Chapter 7: Temporal Data Management

OSTBAYERISCHE TECHNISCHE HOCHSCHULE REGENSBURG

Motivation

Databases usually focus on the current state of the real world.

```
SELECT price FROM products WHERE productno = 29;
UPDATE price FROM products SET price = 4.99 WHERE productno = 29;
```

Information about previous states are no longer available.

Other application scenarios:

- For auditing purposes, a bank needs access to all changes in customer data.
- An insurance company needs to store the fee for specific time periods.
- Predictive-analytics algorithms analyze the past to predict the future.
- A travel agency wants to detect inconsistencies, e.g. the time period of a car rental in Rome overlaps with a hotel reservation in New York.

In some of the scenarios shown here, we need access to the data from the past (historical product prices, customer data, ...) and some examples store also data about the future (insurance fee for next year, hotel reservations, ...).

Temporalization Approaches

Timestamp-based modelling approach:
add two additional columns representing the time period when a record was true

```
CREATE TABLE products (pnr INT, description VARCHAR(100),
price DECIMAL(18,2), valid_from DATE DEFAULT CURRENT_TIME,
                     valid_until DATE DEFAULT '9999-12-31');
```

```
CREATE VIEW products_current AS SELECT * FROM products
WHERE current_time >= valid_from AND current_time < valid_until;
```

Challenges:

- How to interpret time intervals (open/closed)?
- How to write temporal queries?
- How to avoid overlapping time intervals?
- How to add real temporal support on top of the DBMS?
  ⇒ Triggers, stored procedures, constraints.

The table shown here does not have a primary key. (pnr, valid_from) would be better than nothing but it would be desirable if we could somehow avoid that the time periods overlap.

SQL:2011 - Temporal Support

Time Periods

defined by two DATE or TIMESTAMP columns: [start-col, end-col)

```
CREATE TABLE t (d1 DATE, d2 DATE, PERIOD FOR date_period(d1, d2));
```

Application-time period tables

allow temporal queries over periods

```
SELECT * FROM t WHERE date_period CONTAINS DATE '2021-06-01';
```

System-versioned tables

allow historical queries

```
SELECT * FROM t FOR SYSTEM TIME AS OF TIMESTAMP '2021-06-01 14:00';
```

Bitemporal tables

Time periods are defined by a closed-open interval. The start-col is included in the interval, the end-col is not. Bitemporal tables are system-versioned and application-time period tables in combination.

Time Periods

`PERIOD FOR period_name(start_col, end_col)`

```
CREATE TABLE products (pnr INT, description VARCHAR(100),
price DECIMAL(18,2), valid_from DATE, valid_until DATE,
PERIOD FOR date_period(valid_from, valid_until),
UNIQUE(pnr, date_period WITHOUT OVERLAPS));
```

pnr	description	price	valid_from	valid_until
17	chocolate bar	0.89	2021-01-01	2021-07-01
17	chocolate bar	0.99	2021-07-01	9999-12-31

Time periods in MariaDB

MariaDB does not support primary keys on periods ⇒ use `UNIQUE`

The SQL Standard did not introduce a new data type for time periods. Instead, periods are defined over two DATE or TIMESTAMP columns. The period starts at >= start_col and ends at < end_col. In the SQL Standard, periods can be part of the primary key. There, the `WITHOUT OVERLAPS` clause is introduced to guarantee that each point in time is never in more than one period. MariaDB does not support time periods within the primary key. Here, often an additional unique ID column is introduces as the primary key.

Operations on Time Periods

As specified in the SQL Standard (not supported by MariaDB).

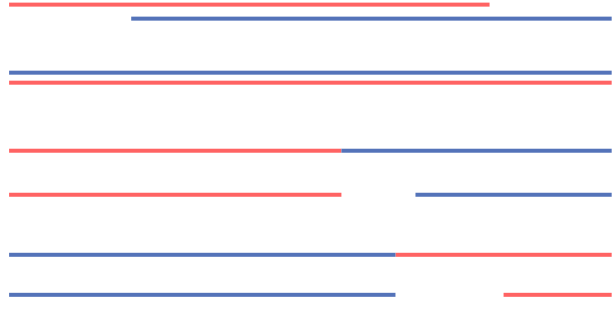
CONTAINS

```
SELECT price FROM products WHERE pnr = 17 AND
date_period CONTAINS '2021-06-29';
```

```
SELECT price FROM products WHERE pnr = 17 AND
date_period CONTAINS PERIOD ('2021-05-01', '2021-05-15');
```

Other operators (based on Allen's Operators)

- p1 OVERLAPS p2
- p1 EQUALS p2
- p1 [IMMEDIATELY] PRECEDES p2
- p1 [IMMEDIATELY] SUCCEEDS p2



Application-Time Period Tables

Application (user) has to set/update valid time periods.

INSERT

```
INSERT INTO products (pnr,description,price,valid_from,valid_until)
VALUES (17, 'chocolate bar', 0.89, '2021-01-01', '2021-07-01');
```

UPDATE

```
UPDATE products
FOR PORTION OF date_period
FROM '2021-02-01' TO '2021-03-01'
SET price = 0.50 WHERE pnr = 17;
```

price	valid_from	valid_until
0.89	2021-01-01	2021-02-01
0.5	2021-02-01	2021-03-01
0.89	2021-03-01	2021-07-01
0.99	2021-07-01	9999-12-31

DELETE

```
DELETE FROM products
FOR PORTION OF date_period FROM '2021-04-15' TO '2021-04-20';
```

Try out at https://bit.ly/mapptime

Application Time: Foreign Keys

As specified by the SQL Standard (not supported by MariaDB)

```
CREATE TABLE subscriptions (custno INT, pnr INT,
valid_from DATE, valid_until DATE,
PERIOD FOR term(valid_from, valid_until),
FOREIGN KEY(pnr, PERIOD term)
REFERENCES products(pnr, PERIOD date_period),
PRIMARY KEY(custno, pnr, term WITHOUT OVERLAPS));
```

subscriptions

custno	pnr	valid_from	valid_until
1	17	2021-04-01	2021-10-01
2	17	2021-09-01	2021-10-01

products

price	valid_from	valid_until
0.89	2021-01-01	2021-07-01
0.99	2021-07-01	9999-12-31

Another table, subscriptions, is also an application-time period table. It stores, that customer 1 has a subscription from April to the end of September and wants to receive a chocolate bar every month. The foreign key ensures referential integrity so that for each subscription, there must be product data which is valid for the full period. For example, we could not insert a subscription from the year 2020.

Temporal Joins

subscriptions

custno	pnr	valid_from	valid_until
1	17	2021-04-01	2021-10-01
2	17	2021-09-01	2021-10-01

products

price	valid_from	valid_until
0.89	2021-01-01	2021-07-01
0.99	2021-07-01	9999-12-31

```
SELECT S.custno, P.*
FROM subscriptions S, products P
WHERE S.pnr = P.pnr AND S.S.term OVERLAPS P.date_period;
```

custno	pnr	description	price	valid_from	valid_until
1	17	chocolate bar	0.89	2021-01-01	2021-07-01
1	17	chocolate bar	0.99	2021-07-01	9999-12-31
2	17	chocolate bar	0.99	2021-07-01	9999-12-31

Without the OVERLAPS operator, we can do the temporal join with `S.valid_from < P.valid_until AND S.valid_until > P.valid_from`

System-versioned Tables

(Only) the DBMS provides start and end time for periods.
All operations within a transaction should use the same timestamp.

```
CREATE TABLE customers(
 custno INT PRIMARY KEY, email VARCHAR(100),
 sys_start TIMESTAMP(6) GENERATED ALWAYS AS ROW START,
 sys_end TIMESTAMP(6) GENERATED ALWAYS AS ROW END,
 PERIOD FOR SYSTEM_TIME(sys_start, sys_end)
) WITH SYSTEM VERSIONING;
```

MariaDB also supports a simplified syntax:

```
CREATE TABLE customers(custno INT PRIMARY KEY, email VARCHAR(100))
WITH SYSTEM VERSIONING;
```

When using MariaDB's short syntax for creating system-versioned tables, the start-time and end-time columns are hidden from the user.

System-versioned Tables

Each modification results in an additional tuple representing the previous row state.

INSERT

Start timestamp: current timestamp; end timestamp: max. TIMESTAMP value

```
INSERT INTO customers (custno, email) VALUES (5, 'peter@example.org');
```

custno	email	sys_start	sys_end
1	peter@example.com	2021-05-01 14:00:00	9999-12-31 23:59:59

UPDATE / DELETE

Operate on current row and insert historical system row.

```
UPDATE customers SET email = 'peter@example.com' WHERE custno = 5;
```

```
DELETE FROM customers WHERE custno = 5;
```

INSERT, UPDATE, DELETE work in the same way as for normal tables. The historical states are automatically saved in the background.

System-versioned Tables

SELECT

By default: only retrieve the current rows.

```
SELECT * FROM customers;  -- empty result; we deleted the customer
```

Historical query with `FOR SYSTEM TIME` clause:

```
SELECT * FROM customers FOR SYSTEM_TIME AS OF '2021-06-01 14:00:00';
```

custno	email	sys_start	sys_end
1	peter@example.com	2021-05-20 11:00:00	2021-06-01 15:00:00

```
SELECT * FROM customers FOR SYSTEM_TIME
FROM '2021-01-01 00:00:00' TO '2021-06-01 14:00:00';
```

custno	email	sys_start	sys_end
1	peter@example.org	2021-05-01 14:00:00	2021-05-01 11:00:00
1	peter@example.com	2021-05-20 11:00:00	2021-06-01 15:00:00

Bitemporal Tables

Application time + system-versioned tables combined.

```
CREATE TABLE products (pnr INT, description VARCHAR(100),
 price DECIMAL(18,2), valid_from DATE, valid_until DATE,
 PERIOD FOR date_period(valid_from, valid_until),
 UNIQUE(pnr, date_period WITHOUT OVERLAPS),
 sys_start TIMESTAMP(6) GENERATED ALWAYS AS ROW START,
 sys_end TIMESTAMP(6) GENERATED ALWAYS AS ROW END,
 PERIOD FOR SYSTEM_TIME(sys_start, sys_end))
WITH SYSTEM VERSIONING;
```

What was on February 1st the planned price that the chocolate bar will have on February 15th?

```
SELECT * FROM products FOR SYSTEM_TIME AS OF '2021-02-01 14:00:00'
WHERE pnr = 17
AND '2021-02-15' >= valid_from AND '2021-02-15' < valid_until;
```

Summary

- SQL:2011/Temporal
- time Periods: `PERIOD FOR ...(t1, t2)`
- operators on time periods: contains, overlaps, ...
- application-time period tables: UPDATE, DELETE, Temporal Joins
- system-versioned tables: `WITH SYSTEM VERSIONING`
- Historical queries: `FOR SYSTEM_TIME ...`
- Bitemporal tables