

Wersjonowanie tabel

(w oparciu o MariaDB)

<https://mariadb.com/kb/en/system-versioned-tables/>

Wersjonowanie tabel

Wersjonowanie tabel - mechanizm historii rekordów w tabelach - inaczej, historii przechowywanych danych w tabelach.

Wersjonowanie istnieje w szbd MariaDB od wersji 10.3.4.

Technicznie, dodawane są ukryte (dla podstawowych zapytań) kolumny: `row_start` i `row_end` przechowujące daty i dokładny czas rekordu, a także adnotacja do klucza głównego. Dzięki temu można w zapytaniu `select` w prosty sposób pobrać dane, które zostały usunięte lub zmodyfikowane w przeszłości, z określeniem np. przedziału dat.

Wersjonowanie w każdej chwili można dodać do tabeli lub usunąć. W razie konieczności edycji (`alter`) struktury wersjonowanej tabeli (np. dodanie kolumny), należy polecenie DDL poprzedzić ustawieniem zmiennej zabezpieczającej przed skutkiem zmiany struktury:

```
set @@system_versioning_alter_history= true;
```

Wersjonowanie tabel

Wersjonowanie tabel - przykład działania cz. 1

```
CREATE TABLE `uzytkownicy` (  
  `ID_uzytkownika` int(100) unsigned NOT NULL AUTO_INCREMENT,  
  `imie` varchar(100) NOT NULL,  
  `nazwisko` varchar(100) NOT NULL,  
  `data_urodzenia` date DEFAULT NULL,  
  PRIMARY KEY (`ID_uzytkownika`)  
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

```
mysql> alter table uzytkownicy add system versioning;  
Query OK, 0 rows affected  
Records: 0 Duplicates: 0 Warnings: 0
```


Dodawanie wersjonowania do
tabeli 'uzytkownicy'

Wersjonowanie tabel

Wersjonowanie tabel - przykład działania cz. 2

```
CREATE TABLE `uzytkownicy` (  
  `ID_uzytkownika` int(100) unsigned NOT NULL AUTO_INCREMENT,  
  `imie` varchar(100) NOT NULL,  
  `nazwisko` varchar(100) NOT NULL,  
  `data_urodzenia` date DEFAULT NULL,  
  PRIMARY KEY (`ID_uzytkownika`)  
) ENGINE=InnoDB DEFAULT CHARSET=utf8 WITH SYSTEM VERSIONING;
```

Tabela z obsługą
wersjonowania



Wersjonowanie tabel

Wersjonowanie tabel - przykład działania cz. 3

```
mysql> select * from uzytkownicy;
```

ID_uzytkownika	imie	nazwisko	data_urodzenia
1	Jan	Kowalski	2020-01-03
2	Anna	Nowak	2021-01-04

```
2 rows in set
```

```
mysql> select *, row_start, row_end from uzytkownicy;
```

ID_uzytkownika	imie	nazwisko	data_urodzenia	row_start	row_end
1	Jan	Kowalski	2020-01-03	2021-01-03 22:23:42.206007	2038-01-19 04:14:07.999999
2	Anna	Nowak	2021-01-04	2021-01-03 22:24:04.781811	2038-01-19 04:14:07.999999

```
2 rows in set
```

Wynik zapytania na tabeli wersjonowanej można poszerzyć o ukryte kolumny: row_start i row_end

Wersjonowanie tabel

Wersjonowanie tabel - przykład działania cz. 4

```
mysql> delete from uzytkownicy where ID_uzytkownika = 2;  
Query OK, 1 row affected
```

```
mysql> select *, row_start, row_end from uzytkownicy;
```

ID_uzytkownika	imie	nazwisko	data_urodzenia	row_start	row_end
1	Jan	Kowalski	2020-01-03	2021-01-03 22:23:42.206007	2038-01-19 04:14:07.999999

```
1 row in set
```

```
mysql> select *, row_start, row_end from uzytkownicy for system_time all;
```

ID_uzytkownika	imie	nazwisko	data_urodzenia	row_start	row_end
1	Jan	Kowalski	2020-01-03	2021-01-03 22:23:42.206007	2038-01-19 04:14:07.999999
2	Anna	Nowak	2021-01-04	2021-01-03 22:24:04.781811	2021-01-03 22:29:27.943766

```
2 rows in set
```

Użycie klauzuli „for system time all” wyświetla całą historię tabeli. Do każdego z rekordów historii można odwołać się poprzez znane polecenia, np. „where”, „group by”, „order by”, itp.

Wersjonowanie tabel

Wersjonowanie tabel - przykład działania cz. 5

```
mysql> insert into uzytkownicy values (2, 'Anna', 'Nowak', '2021-01-04');  
Query OK, 1 row affected
```

```
mysql> select *, row_start, row_end from uzytkownicy for system_time all;
```

ID_uzytkownika	imie	nazwisko	data_urodzenia	row_start	row_end
1	Jan	Kowalski	2020-01-03	2021-01-03 22:23:42.206007	2038-01-19 04:14:07.999999
2	Anna	Nowak	2021-01-04	2021-01-03 22:24:04.781811	2021-01-03 22:29:27.943766
2	Anna	Nowak	2021-01-04	2021-01-03 22:33:00.255126	2038-01-19 04:14:07.999999

3 rows in set

```
mysql> select *, row_start, row_end from uzytkownicy for system_time from '2021-01-03 21:00' to '2021-01-03 22:25';
```

ID_uzytkownika	imie	nazwisko	data_urodzenia	row_start	row_end
1	Jan	Kowalski	2020-01-03	2021-01-03 22:23:42.206007	2038-01-19 04:14:07.999999
2	Anna	Nowak	2021-01-04	2021-01-03 22:24:04.781811	2021-01-03 22:29:27.943766

2 rows in set

```
mysql> select *, row_start, row_end from uzytkownicy for system_time from '2021-01-03 21:00' to now();
```

ID_uzytkownika	imie	nazwisko	data_urodzenia	row_start	row_end
1	Jan	Kowalski	2020-01-03	2021-01-03 22:23:42.206007	2038-01-19 04:14:07.999999
2	Anna	Nowak	2021-01-04	2021-01-03 22:24:04.781811	2021-01-03 22:29:27.943766
2	Anna	Nowak	2021-01-04	2021-01-03 22:33:00.255126	2038-01-19 04:14:07.999999

3 rows in set

Podstawy baz danych / Artur Miewiarowski

Różne możliwości odwoływania się poprzez daty do historii rekordów

Wersjonowanie tabel

Wersjonowanie tabel - przykład działania cz. 6

```
mysql> select *, row_start, row_end from uzytkownicy for system_time from now() - interval 1 hour to now() - interval 15 minute;
```

ID_uzytkownika	imie	nazwisko	data_urodzenia	row_start	row_end
1	Jan	Kowalski	2020-01-03	2021-01-03 22:23:42.206007	2038-01-19 04:14:07.999999
2	Anna	Nowak	2021-01-04	2021-01-03 22:24:04.781811	2021-01-03 22:29:27.943766

2 rows in set

```
mysql> select *, row_start, row_end from uzytkownicy for system_time all where nazwisko = 'Nowak';
```

ID_uzytkownika	imie	nazwisko	data_urodzenia	row_start	row_end
2	Anna	Nowak	2021-01-04	2021-01-03 22:24:04.781811	2021-01-03 22:29:27.943766
2	Anna	Nowak	2021-01-04	2021-01-03 22:33:00.255126	2038-01-19 04:14:07.999999

2 rows in set

Wersjonowanie tabel

Wersjonowanie tabel - błędne próby odwołania do historii

```
mysql> select *, row_start, row_end from uzytkownicy where row_start='2021-01-03 22:33:00.255126';
```

ID_uzytkownika	imie	nazwisko	data_urodzenia	row_start	row_end
2	Anna	Nowak	2021-01-04	2021-01-03 22:33:00.255126	2038-01-19 04:14:07.999999

```
1 row in set
```

```
mysql> select *, row_start, row_end from uzytkownicy where row_start='2021-01-03 22:24:04.781811';  
Empty set
```

```
mysql> select *, row_start, row_end from uzytkownicy where row_start like '2021-01-03%';
```

ID_uzytkownika	imie	nazwisko	data_urodzenia	row_start	row_end
1	Jan	Kowalski	2020-01-03	2021-01-03 22:23:42.206007	2038-01-19 04:14:07.999999
2	Anna	Nowak	2021-01-04	2021-01-03 22:33:00.255126	2038-01-19 04:14:07.999999

```
2 rows in set
```

Poprzez powyższe odwołania operujemy wyłącznie na aktualnych rekordach

Wersjonowanie tabel

Zapamiętywanie historii rekordów sprawia, że tabela jest fizycznie większa, niż przy wyłączonej opcji wersjonowania. Pociąga to za sobą dłuższe oczekiwanie na odpowiedzi na zapytania, przy bardzo dużej liczbie rekordów.

Rozwiązaniem tej niedogodności jest użycie mechanizmu partycjonowania tabel - czyli podziału tabeli fizycznie na części, a następnie pogrupowaniu zestawu przechowywanych danych względem utworzonych partycji.

```
CREATE TABLE t (x INT) WITH SYSTEM  
VERSIONING  
PARTITION BY SYSTEM_TIME (  
    PARTITION p_hist HISTORY,  
    PARTITION p_cur CURRENT  
);
```

Wersjonowanie tabel

Wersjonowanie tabel - optymalizacja poprzez partycjonowanie

```
mysql>
  alter table uzytkownicy partition by system_time (
    PARTITION p_historia HISTORY,
    PARTITION p_aktualnie CURRENT
  )
;
```

Query OK, 3 rows affected

Records: 3 Duplicates: 0 Warnings: 0

```
mysql> SELECT *, row_start, row_end FROM uzytkownicy PARTITION (p_aktualnie);
```

ID_uzytkownika	imie	nazwisko	data_urodzenia	row_start	row_end
1	Jan	Kowalski	2020-01-03	2021-01-03 22:23:42.206007	2038-01-19 04:14:07.999999
2	Anna	Nowak	2021-01-04	2021-01-03 22:33:00.255126	2038-01-19 04:14:07.999999

2 rows in set

```
mysql> SELECT *, row_start, row_end FROM uzytkownicy PARTITION (p_historia);
```

ID_uzytkownika	imie	nazwisko	data_urodzenia	row_start	row_end
2	Anna	Nowak	2021-01-04	2021-01-03 22:24:04.781811	2021-01-03 22:29:27.943766

1 row in set

Wersjonowanie tabel

Wersjonowanie tabel - optymalizacja poprzez partycjonowanie

```
mysql> explain partitions select * from uzytkownicy;
```

id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	Extra
1	SIMPLE	uzytkownicy	p_aktualnie	ALL	NULL	NULL	NULL	NULL	2	Using where

```
1 row in set
```

```
mysql> explain partitions select * from uzytkownicy for system_time all;
```

id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	Extra
1	SIMPLE	uzytkownicy	p_historia,p_aktualnie	ALL	NULL	NULL	NULL	NULL	3	

```
mysql> explain partitions select * from uzytkownicy for system_time from '2021-01-03 22:25' to '2021-01-03 22:29';
```

id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	Extra
1	SIMPLE	uzytkownicy	p_historia,p_aktualnie	ALL	NULL	NULL	NULL	NULL	3	Using where

```
1 row in set
```

```
mysql> explain partitions select * from uzytkownicy for system_time from '2010-01-03 22:25' to '2011-01-03 22:29';
```

id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	Extra
1	SIMPLE	uzytkownicy	p_historia,p_aktualnie	ALL	NULL	NULL	NULL	NULL	3	Using where

```
1 row in set
```

Odwołanie się do aktualnych danych sprawia przeszukanie wyłącznie partycji p_aktualne (tj. z aktualnymi rekordami).

Każde odwołanie się do historii rekordów (czyli wywołanie przeszukania po datach z czasem) przeszukuje wszystkie partycje.

Wersjonowanie tabel

Wersjonowanie tabel - optymalizacja poprzez partycjonowanie

Więcej możliwości związanych z partycjonowaniem pod kątem wersjonowania znajduje się pod tym linkiem:

<https://mariadb.com/kb/en/system-versioned-tables/#storing-the-history-separately>

Wersjonowanie tabel

Wersjonowanie tabel - usuwanie starych danych

1)
`delete history from uzytkownicy before system_time '2020-01-01';`

--Wymaga specjalnych uprawnień - np. root'a

2)
`alter table uzytkownicy DROP PARTITION p_historia;`

--Wymaga pozostania minimum jednej partycji z historią

3)
`ALTER TABLE uzytkownicy DROP SYSTEM VERSIONING;`
`ALTER TABLE uzytkownicy ADD SYSTEM VERSIONING;`