

# Partycjonowanie tabel

(w oparciu o MariaDB)

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

# Czym jest partycjonowanie tabel

- ▶ Partycjonowanie jest fizycznym podziałem tabeli na części według odpowiedniego klucza i mechanizmu.
- ▶ Fizycznie na dysku, każda z partycji jest osobnym plikiem z własnymi indeksami. Pliki mogą zostać rozmieszczone na różnych nośnikach lub partycjach dysku.
- ▶ Celem partycjonowania jest przyspieszenie wykonywania zapytań na tabeli poprzez odniesienie się w zapytaniu do danych będących kluczem podziału. Dodatkowo umożliwia przechowywanie partycji zawierających dane historyczne na osobnych dyskach o szczególnym przeznaczeniu.

# Czym jest partycjonowanie tabel

```
c:\Program Files\MariaDB 10.5\data\nowa@0020baza>dir
21:54          2 804 kierunki.frm
21:54       147 456 kierunki.ibd
21:03       98 304 logi#p#mniejsze_od_2021.ibd
21:03       98 304 logi#p#mniejsze_od_2022.ibd
21:03       98 304 logi#p#mniejsze_od_2023.ibd
21:02       98 304 logi#p#wieksze_od_2022.ibd
21:02       1 397 logi.frm      << definicja tabeli
21:02       104 logi.par      << definicja partycji
23:09       3 926 prace_dyplomowe.frm
23:23      163 840 prace_dyplomowe.ibd
```

# Czym jest partycjonowanie tabel

- ▶ Partycjonowanie powinno mieć miejsce na tabelach, które docelowo będą (ew. już mają) bardzo dużo rekordów (tutaj mówimy o milionach wierszy) i do których jest częsty dostęp.
- ▶ Dodatkowo dane w tabelach powinny być wyraźnie podzielone na stare, starsze, nowe, nowsze, ewentualnie podzielone według wyraźnych przedziałów liczbowych.

# Dostępność partycjonowania w systemie

- ▶ polecenie `show plugins - partition active`
- ▶ polecenie `show variable like '%partition%'` - w starszych wersjach MySQL/MariaDB
- ▶ Jeżeli szbd nie obsługuje partycjonowania, to może oznaczać, że nie został zainstalowany w systemie operacyjnym z taką możliwością lub został uruchomiony z parametrami wyłączającymi obsługę partycjonowania, tj.:

`--skip-partition`

`--disable-partition`

`--partition=OFF`

# Dostępność partycjonowania w systemie

## ► polecenie `show plugins - partition active`

```
mariadb> show plugins;
```

Name	Status	Type	Library	License
user_variables	ACTIVE	INFORMATION SCHEMA	NULL	GPL
THREAD_POOL_GROUPS	ACTIVE	INFORMATION SCHEMA	NULL	GPL
THREAD_POOL_QUEUES	ACTIVE	INFORMATION SCHEMA	NULL	GPL
THREAD_POOL_STATS	ACTIVE	INFORMATION SCHEMA	NULL	GPL
THREAD_POOL_WAITS	ACTIVE	INFORMATION SCHEMA	NULL	GPL
<b>partition</b>	<b>ACTIVE</b>	<b>STORAGE ENGINE</b>	<b>NULL</b>	<b>GPL</b>

# Podział tabeli na partycje

- ▶ Partycje można tworzyć podczas wykonywania polecenia "create table" tworzącego tabelę.
- ▶ jak również w ramach polecenie "alter table" zmieniającego strukturę tabeli, gdzie możemy: dodać partycje do istniejącej tabeli, usunąć wybrane partycje, zreorganizować je według nowych parametrów.

```
CREATE TABLE `logi` (  
  `ID_logi` INT ( 10 ) UNSIGNED NOT NULL AUTO_INCREMENT,  
  `ID_uzytkownicy` INT ( 10 ) UNSIGNED DEFAULT NULL,  
  `dt` datetime NOT NULL,  
  `IP` VARCHAR ( 20 ) DEFAULT NULL,  
  PRIMARY KEY ( `ID_logi`, `dt` ) USING BTREE  
) ENGINE = INNODB DEFAULT CHARSET = utf8 PARTITION BY RANGE (  
  YEAR ( `dt` ))(  
  PARTITION `mniejwsze_od_2021`  
  VALUES  
    LESS THAN ( 2021 ) ENGINE = INNODB,  
  PARTITION `mniejwsze_od_2022`  
  VALUES  
    LESS THAN ( 2022 ) ENGINE = INNODB,  
  PARTITION `mniejwsze_od_2023`  
  VALUES  
    LESS THAN ( 2023 ) ENGINE = INNODB,  
  PARTITION `wieksze_od_2022`  
  VALUES  
    LESS THAN MAXVALUE ENGINE = INNODB  
  );
```

# Pruning (przycinanie/okrajanie) i selekcja

- ▶ partition pruning - oznacza ujęcie kolumn-kluczy w klauzuli where zapytania, co sprawia, że optymalizator zapytania wie, w której partycji znajdują się poszukiwane rekordy. W takim przypadku pozostałe partycje nie będą przeszukiwane
- ▶ partition selection oznacza umieszczenie klauzuli PARTITION po nazwie tabeli w celu określenia, której partycji ma użyć optymalizator, ponieważ w klauzuli where nie ujęto kolumny-klucza.



# Pruning

```
mariadb> explain partitions select * from logi where dt < '2022-01-01';
```

```
+----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
-----+
| id | select_type | table | partitions | type | possible_keys | key | key_len | ref | rows |
Extra |
+----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
-----+
| 1 | SIMPLE | logi | mniejsze_od_2021,mniejsze_od_2022 | ALL | NULL | NULL | NULL | NULL | 3 |
Using where |
+----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
-----+

1 row in set (0.03 sec)
```

```
mariadb> explain partitions select * from logi where dt <= '2022-01-01';
```

```
+----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
----+-----+-----+
| id | select_type | table | partitions | type | possible_keys | key | key_len | ref | rows |
ref | rows | Extra |
+----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
----+-----+-----+
| 1 | SIMPLE | logi | mniejsze_od_2021,mniejsze_od_2022,mniejsze_od_2023 | ALL | NULL | NULL | NULL | NULL |
NULL | 4 | Using where |
+----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
----+-----+-----+

1 row in set (0.04 sec)
```

# Selekcja

```
mariadb> select * from logi PARTITION(mniejsze_od_2021);
```

ID_logi	ID_uzytkownicy	dt	IP
1	1	2020-01-01 00:00:00	NULL
4	1	2019-01-01 00:00:00	NULL

```
2 rows in set (0.02 sec)
```

```
mariadb> select * from logi PARTITION(mniejsze_od_2022);
```

ID_logi	ID_uzytkownicy	dt	IP
2	1	2021-01-01 00:00:00	NULL


```
1 row in set (0.02 sec)
```

# Ograniczenia w partycjonowaniu

Wybrane ograniczenia w partycjonowaniu w MariaDB:

- ▶ Każda tabela może posiadać maksimum 8192 partycje
- ▶ Partycjonowanie jest możliwe na tabeli wyłącznie wtedy, gdy dany silnik obsługuje partycjonowanie
- ▶ Wszystkie partycje danej tabeli muszą posiadać ten sam silnik

```
mariadb> CREATE TABLE `logi_engines` (  
  `ID_logi` int(10) unsigned NOT NULL AUTO_INCREMENT,  
  `ID_uzytkownicy` int(10) unsigned DEFAULT NULL,  
  `dt` datetime NOT NULL,  
  `IP` varchar(20) DEFAULT NULL,  
  PRIMARY KEY (`ID_logi`,`dt`) USING BTREE  
) ENGINE=InnoDB AUTO_INCREMENT=5 DEFAULT CHARSET=utf8  
  PARTITION BY RANGE (`year`(`dt`))  
(PARTITION `mniej_sze_od_2021` VALUES LESS THAN (2021) ENGINE = MyISAM,  
  PARTITION `mniej_sze_od_2022` VALUES LESS THAN (2022) ENGINE = InnoDB,  
  PARTITION `mniej_sze_od_2023` VALUES LESS THAN (2023) ENGINE = InnoDB,  
  PARTITION `wieksze_od_2022` VALUES LESS THAN MAXVALUE ENGINE = InnoDB);  
1497 -  
The mix of handlers in the partitions is not allowed in this version of  
MariaDB  
mariadb>
```




# Ograniczenia w partycjonowaniu

Wybrane ograniczenia w partycjonowaniu w MariaDB:

- Tabela partycjonowana nie może zawierać klucza obcego

```
mariadb> ALTER TABLE `nowa baza`.`logi`  
ADD FOREIGN KEY (`ID_uzytkownicy`) REFERENCES `nowa baza`.`uzytkownicy`  
(`ID_uzytkownicy`) ON DELETE CASCADE ON UPDATE CASCADE;  
1506 - Partitioned tables do not support FOREIGN KEY  
mariadb>
```



# Ograniczenia w partycjonowaniu

Wybrane ograniczenia w partycjonowaniu w MariaDB:

- ▶ Jeżeli tabela posiada indeksy unikalne (PK, unique), to wszystkie kolumny wchodzące w skład klucza partycjonowania muszą zostać ujęte w indeks

```
CREATE TABLE `logi_uni` (  
  `ID_logi` int(10) unsigned NOT NULL AUTO_INCREMENT,  
  `ID_uzytkownicy` int(10) unsigned DEFAULT NULL,  
  `dt` datetime NOT NULL,  
  `IP` varchar(20) DEFAULT NULL,  
  PRIMARY KEY (`ID_logi`) USING BTREE  
) ENGINE=InnoDB AUTO_INCREMENT=5 DEFAULT CHARSET=utf8  
PARTITION BY RANGE (`year`(`dt`))  
(PARTITION `mniejsze_od_2021` VALUES LESS THAN (2021) ENGINE = InnoDB,  
PARTITION `mniejsze_od_2022` VALUES LESS THAN (2022) ENGINE = InnoDB,  
PARTITION `mniejsze_od_2023` VALUES LESS THAN (2023) ENGINE = InnoDB,  
PARTITION `wieksze_od_2022` VALUES LESS THAN MAXVALUE ENGINE = InnoDB  
);
```

# Ograniczenia w partycjonowaniu

Wybrane ograniczenia w partycjonowaniu w MariaDB

► Prawidłowy kod:

```
CREATE TABLE `logi_uni` (  
  `ID_logi` int(10) unsigned NOT NULL AUTO_INCREMENT,  
  `ID_uzytkownicy` int(10) unsigned DEFAULT NULL,  
  `dt` datetime NOT NULL,  
  `IP` varchar(20) DEFAULT NULL,  
  PRIMARY KEY (`ID_logi`, `dt`) USING BTREE  
) ENGINE=InnoDB AUTO_INCREMENT=5 DEFAULT CHARSET=utf8  
  PARTITION BY RANGE (`year`(`dt`))  
(PARTITION `mniejsze_od_2021` VALUES LESS THAN (2021) ENGINE = InnoDB,  
PARTITION `mniejsze_od_2022` VALUES LESS THAN (2022) ENGINE = InnoDB,  
PARTITION `mniejsze_od_2023` VALUES LESS THAN (2023) ENGINE = InnoDB,  
PARTITION `wieksze_od_2022` VALUES LESS THAN MAXVALUE ENGINE = InnoDB  
);  
Query OK, 0 rows affected (0.05 sec)
```

# Podział tabeli na partycje - wybrane typy partycji

- ▶ **RANGE** - partycjonowanie z uwzględnieniem przedziałów liczbowych (int) zdefiniowanych przez programistę
- ▶ **LIST** - partycjonowanie z uwzględnieniem liczb (int) zawartych na liście liczb zdefiniowanej przez programistę
- ▶ **HASH** - równomierne rozłożenie rekordów w partycjach przez szbd po wartościach liczbowych (int)
- ▶ **RANGE COLUMNS** - partycjonowanie po wartościach typu: int, string, date i datetime. Możliwość uwzględnienia wielu kolumn, brak możliwości stosowania wyrażeń
- ▶ **LIST COLUMNS** - j.w., tyle, że wartości podziału zdefiniowane są w postaci listy wartości
- ▶ **SYSTEM\_TIME** - omówione w ramach wersjonowania tabel

# Podział tabeli na partycje

## RANGE / 1

- partycjonowanie z uwzględnieniem przedziałów liczbowych (int) zdefiniowanych przez programistę

```
CREATE TABLE `logi` (  
  `ID_logi` int(10) unsigned NOT NULL AUTO_INCREMENT,  
  `ID_uzytkownicy` int(10) unsigned DEFAULT NULL,  
  `dt` datetime NOT NULL,  
  `IP` varchar(20) DEFAULT NULL,  
  PRIMARY KEY (`ID_logi`,`dt`) USING BTREE  
) ENGINE=InnoDB AUTO_INCREMENT=5 DEFAULT CHARSET=utf8  
PARTITION BY RANGE (`year`(`dt`))  
(PARTITION `mniej_od_2021` VALUES LESS THAN (2021) ENGINE = InnoDB,  
PARTITION `mniej_od_2022` VALUES LESS THAN (2022) ENGINE = InnoDB,  
PARTITION `mniej_od_2023` VALUES LESS THAN (2023) ENGINE = InnoDB,  
PARTITION `wieksze_od_2022` VALUES LESS THAN MAXVALUE ENGINE = InnoDB  
);  
Query OK, 0 rows affected (0.05 sec)
```

```
mariadb> explain partitions select * from logi where dt < '2022-01-01';
```

id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	Extra
1	SIMPLE	logi	mniejszy_od_2021,mniejszy_od_2022	ALL	NULL	NULL	NULL	NULL	3	Using where

```
1 row in set (0.03 sec)
```



# Podział tabeli na partycje

## RANGE / 1

Table View Function Event User Query Backup Automation Model Charts

Objects root2@mariadb

Save

Fields Indexes Foreign Keys

Engine:

Tablespace:

Storage:

Character set:

Collation:

Auto Increment:

Row Format:

Avg. Row Length:

Max Rows:

Key Block Size:

Data Directory:

Index Directory:

Stats Auto Recalc:

Stats Persistent:

Stats Sample Pages:

☐ Encryption

Partition

Partition

Partition By: RANGE year('dt')

Partitions: 4

Subpartition By:

Subpartitions: 0

Manually Create: ☒ Partition ☒ Subpartition

Partition Definition

mniejszy_od_2021	2021
mniejszy_od_2022	2022
mniejszy_od_2023	2023
wieksze_od_2022	MAXVALUE

Name: mniejsze\_od\_2021

Values: 2021

Engine: InnoDB

Data Directory:

Index Directory:

Max Rows: 0 Min Rows: 0

Tablespace:

Node Group:

Comment:

OK

# Podział tabeli na partycje

## RANGE / 2

Partition

Partition By: RANGE to\_days(dt)

Partitions: 2

Subpartition By:

Subpartitions: 0

Manually Create: ☒ Partition ☒ Subpartition

Partition Definition

do_2021_05	to_days('2021-06-01')
od_2021_06_do_2021_12_31	to_days('2022-01-01')

Name: od\_2021\_06\_do\_2021\_12\_31

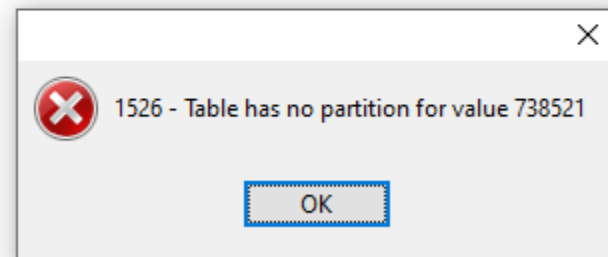
Values: to\_days('2022-01-01')

Engine:

```
ALTER TABLE `nowa baza`.`logi1` PARTITION BY RANGE (to_days(dt))
PARTITIONS 2
(PARTITION `do_2021_05` VALUES LESS THAN (to_days('2021-06-01')) MAX_ROWS = 0 MIN_ROWS = 0 ,
PARTITION `od_2021_06_do_2021_12_31` VALUES LESS THAN (to_days('2022-01-01')) MAX_ROWS = 0 MIN_ROWS = 0 )
;
```

# Podział tabeli na partycje RANGE / 2

ID_logi	ID_uzytkownicy	dt	IP
1		2020-01-01 00:00:00	(Null)
2		2021-05-20 00:00:00	(Null)
3		2021-06-01 00:00:00	(Null)
(Null)		2022-01-01	(Null)



```
mariadb> explain partitions select * from logi1 where dt = '2020-10-01 12:00:35';
```

```
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| id | select_type | table | partitions | type | possible_keys | key | key_len | ref | rows | Extra |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| 1 | SIMPLE | logi1 | do_2021_05 | ALL | NULL | NULL | NULL | NULL | 2 | Using where |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
1 row in set (0.04 sec)
```

```
mariadb> explain partitions select * from logi1 where dt = '2021-10-01 13:00:35';
```

```
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| id | select_type | table | partitions | type | possible_keys | key | key_len | ref | rows | Extra |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| 1 | SIMPLE | logi1 | od_2021_06_do_2021_12_31 | ALL | NULL | NULL | NULL | NULL | 2 | Using where |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
1 row in set (0.04 sec)
```

# Podział tabeli na partycje LIST

- partycjonowanie z uwzględnieniem liczb (int) zawartych na liście liczb zdefiniowanej przez programistę

```
ALTER TABLE `nowa baza`.`logi2` PARTITION BY LIST (year(dt))  
PARTITIONS 4  
(PARTITION `rok_2021_2022_2023` VALUES IN (2021,2022,2023) MAX_ROWS = 0 MIN_ROWS = 0 ,  
PARTITION `rok_2024_2025` VALUES IN (2024,2025) MAX_ROWS = 0 MIN_ROWS = 0 ,  
PARTITION `rok_2026` VALUES IN (2026) MAX_ROWS = 0 MIN_ROWS = 0 ,  
PARTITION `rok_2027_2028_2029_2030` VALUES IN (2027,2028,2029,2030) MAX_ROWS = 0 MIN_ROWS = 0  
)  
;
```

Partition

Partition By: LIST year(dt)

Partitions: 4

Subpartition By:

Subpartitions: 0

Manually Create: ☒ Partition ☒ Subpartition

Partition Definition

rok_2021_2022_2023	2021,2022,2023
rok_2024_2025	2024,2025
rok_2026	2026
rok_2027_2028_2029_2030	2027,2028,2029,2030

Name: rok\_2021\_2022\_2023

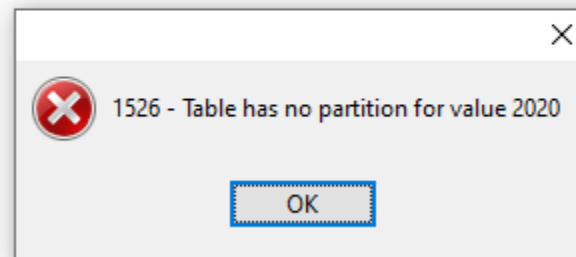
Values: 2021,2022,2023

# Podział tabeli na partycje LIST

- ▶ partycjonowanie z uwzględnieniem liczb (int) zawartych na liście liczb zdefiniowanej przez programistę

```
CREATE TABLE `logi2` (  
  `ID_logi` int(10) unsigned NOT NULL AUTO_INCREMENT,  
  `ID_uzytkownicy` int(10) unsigned DEFAULT NULL,  
  `dt` datetime NOT NULL,  
  `IP` varchar(20) DEFAULT NULL,  
  PRIMARY KEY (`ID_logi`, `dt`) USING BTREE  
) ENGINE=InnoDB DEFAULT CHARSET=utf8  
PARTITION BY LIST (year(`dt`))  
(PARTITION `rok_2021_2022_2023` VALUES IN (2021,2022,2023) ENGINE = InnoDB,  
PARTITION `rok_2024_2025` VALUES IN (2024,2025) ENGINE = InnoDB,  
PARTITION `rok_2026` VALUES IN (2026) ENGINE = InnoDB,  
PARTITION `rok_2027_2028_2029_2030` VALUES IN (2027,2028,2029,2030) ENGINE = InnoDB);
```

ID_logi	ID_uzytkownicy	dt	IP
1	1	2021-01-01 00:00:00	(Null)
(Null)	1	2020-01-01	(Null)



# Podział tabeli na partycje LIST

- partycjonowanie z uwzględnieniem liczb (int) zawartych na liście liczb zdefiniowanej przez programistę

```
mariadb> explain partitions select * from logi2;
```

```
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| id | select_type | table | partitions | type | possible_keys | key | key_len | ref | rows | Extra |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| 1 | SIMPLE | logi2 | rok_2021_2022_2023,rok_2024_2025,rok_2026,rok_2027_2028_2029_2030 | ALL | NULL | NULL | NULL | NULL | 4 | Using where |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
1 row in set (0.02 sec)
```

```
mariadb> explain partitions select * from logi2 where dt between '2021-01-01' and '2021-12-31';
```

```
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| id | select_type | table | partitions | type | possible_keys | key | key_len | ref | rows | Extra |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| 1 | SIMPLE | logi2 | rok_2021_2022_2023 | ALL | NULL | NULL | NULL | NULL | 2 | Using where |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
1 row in set (0.02 sec)
```

```
mariadb> explain partitions select * from logi2 where dt between '2000-01-01' and '2010-12-31';
```

```
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| id | select_type | table | partitions | type | possible_keys | key | key_len | ref | rows | Extra |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| 1 | SIMPLE | NULL | NULL | NULL | NULL | NULL | NULL | NULL | NULL | Impossible WHERE noticed after r
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
1 row in set (0.02 sec)
```

# Podział tabeli na partycje HASH

- ▶ równomierne rozłożenie rekordów w partycjach przez szbd po wartościach liczbowych (int)

```
ALTER TABLE `nowa baza`.`logi3` PARTITION BY HASH (year(dt))  
PARTITIONS 10  
(PARTITION `p0` MAX_ROWS = 0 MIN_ROWS = 0 ,  
PARTITION `p1` MAX_ROWS = 0 MIN_ROWS = 0 ,  
PARTITION `p2` MAX_ROWS = 0 MIN_ROWS = 0 ,  
PARTITION `p3` MAX_ROWS = 0 MIN_ROWS = 0 ,  
PARTITION `p4` MAX_ROWS = 0 MIN_ROWS = 0 ,  
PARTITION `p5` MAX_ROWS = 0 MIN_ROWS = 0 ,  
PARTITION `p6` MAX_ROWS = 0 MIN_ROWS = 0 ,  
PARTITION `p7` MAX_ROWS = 0 MIN_ROWS = 0 ,  
PARTITION `p8` MAX_ROWS = 0 MIN_ROWS = 0 ,  
PARTITION `p9` MAX_ROWS = 0 MIN_ROWS = 0 )  
;
```

The screenshot shows the 'Partition' dialog box in a database management tool. The 'Partition By' dropdown is set to 'HASH' and the 'Partitions' field is set to '10'. The 'Subpartition By' dropdown is empty and the 'Subpartitions' field is set to '0'. The 'Manually Create' section has both 'Partition' and 'Subpartition' checkboxes checked. The 'Partition Definition' list shows partitions p0 through p5. The 'Name' field at the bottom is set to 'p0'.

Partition

Partition By: HASH year(dt)

Partitions: 10

Subpartition By:

Subpartitions: 0

Manually Create: ☒ Partition ☒ Subpartition

Partition Definition

- p0
- p1
- p2
- p3
- p4
- p5

Name: p0

Values:

Engine:

# Podział tabeli na partycje

## RANGE COLUMNS

- partycjonowanie po wartościach typu: int, string, date i datetime. Możliwość uwzględnienia wielu kolumn, brak możliwości stosowania wyrażeń

```
mariadb> ALTER TABLE UZYTKOWNICY2
PARTITION BY RANGE COLUMNS(nazwisko) (
PARTITION a VALUES LESS THAN ('b'),
PARTITION b VALUES LESS THAN ('c'),
PARTITION c VALUES LESS THAN ('d'),
PARTITION d VALUES LESS THAN (maxvalue)
);
```

Query OK, 0 rows affected (0.10 sec)

Records: 0 Duplicates: 0 Warnings: 0

```
mariadb>
```



# Podział tabeli na partycje

## RANGE COLUMNS

```
mariadb> select * from uzytkownicy2;
```

ID_uzytkownicy	login	haslo	imie	nazwisko
14	anna	x	anna	abram
16	333	3	3	3
15	michał	x	michał	nowak

```
3 rows in set (0.02 sec)
```

```
mariadb> explain partitions select * from uzytkownicy2 where nazwisko='abramczyk';
```

id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	Extra
1	SIMPLE	uzytkownicy2	a	ALL	NULL	NULL	NULL	NULL	2	Using where

```
1 row in set (0.02 sec)
```

```
mariadb> explain partitions select * from uzytkownicy2 where nazwisko='3';
```

id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	Extra
1	SIMPLE	uzytkownicy2	a	ALL	NULL	NULL	NULL	NULL	2	Using where

```
1 row in set (0.02 sec)
```

```
mariadb> explain partitions select * from uzytkownicy2 where nazwisko='zebra';
```

id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	Extra
1	SIMPLE	uzytkownicy2	d	ALL	NULL	NULL	NULL	NULL	2	Using where

```
1 row in set (0.02 sec)
```

```
mariadb> select * from uzytkownicy2 partition(a);
```

ID_uzytkownicy	login	haslo	imie	nazwisko
14	anna	x	anna	abram
16	333	3	3	3

Podstawy baz danych / Artur Niewiarowski

```
2 rows in set (0.03 sec)
```

```
mariadb> select* from uzytkownicy2 partition(b);
```

```
Empty set
```

# Reorganizacja partycji

```
CREATE TABLE `logi4` (  
  `ID_logi` int(10) unsigned NOT NULL AUTO_INCREMENT,  
  `ID_uzytkownicy` int(10) unsigned DEFAULT NULL,  
  `dt` datetime NOT NULL,  
  `IP` varchar(20) DEFAULT NULL,  
  PRIMARY KEY (`ID_logi`,`dt`) USING BTREE  
) ENGINE=InnoDB AUTO_INCREMENT=5 DEFAULT CHARSET=utf8  
PARTITION BY RANGE (`year`(`dt`))  
(PARTITION `mniej_sze_od_2021` VALUES LESS THAN (2021) ENGINE = InnoDB,  
PARTITION `mniej_sze_od_2022` VALUES LESS THAN (2022) ENGINE = InnoDB,  
PARTITION `mniej_sze_od_2023` VALUES LESS THAN (2023) ENGINE = InnoDB,  
PARTITION `najnowsze` VALUES LESS THAN MAXVALUE ENGINE = InnoDB);
```

# Reorganizacja partycji

## usuwanie niepotrzebnych partycji

```
mariadb> alter table logi4 drop partition mniejsze_od_2021;  
Query OK, 0 rows affected (0.03 sec)  
Records: 0 Duplicates: 0 Warnings: 0
```

```
mariadb>
```

Partycja usuwa się wraz z danymi

# Reorganizacja partycji

## usuwanie niepotrzebnych danych z partycji

```
mariadb> ALTER TABLE logi4 TRUNCATE PARTITION mniejsze_od_2023;  
Query OK, 0 rows affected (0.03 sec)
```

```
mariadb>
```

# Reorganizacja partycji

## dodawanie nowej partycji na bazie istniejącej, reorganizacja danych

```
ALTER TABLE logi4  
REORGANIZE PARTITION najnowsze INTO (  
PARTITION mniejsze_od_2024 VALUES LESS THAN (2024),  
PARTITION najnowsze VALUES LESS THAN MAXVALUE);
```

Wraz z nadejściem nowego roku tworzymy nową partycję

# Reorganizacja partycji

## dodawanie nowej partycji na bazie istniejącej, reorganizacja danych

Nie możemy dodać nowej partycji dla mniejszych wartości bez reorganizacji istniejącej partycji

```
mariadb> ALTER TABLE logi4 ADD PARTITION (PARTITION  
mniejsze_od_2021 VALUES LESS THAN (2021));
```

1493 -

VALUES LESS THAN value must be strictly increasing  
for each partition

```
ALTER TABLE logi4  
REORGANIZE PARTITION mniejsze_od_2022 INTO (  
PARTITION mniejsze_od_2021 VALUES LESS THAN (2021),  
PARTITION mniejsze_od_2022 VALUES LESS THAN (2022));
```

# Reorganizacja partycji

## dodawanie nowej partycji (przy braku maxvalue)

```
mariadb> ALTER TABLE logi4 ADD PARTITION (PARTITION mniejsze
_od_2025 VALUES LESS THAN (2025));
Query OK, 0 rows affected (0.03 sec)
Records: 0 Duplicates: 0 Warnings: 0
```

Nie możemy dodać nowej partycji bez reorganizacji dla nowszych danych, jeżeli mamy zdefiniowaną partycję z maxvalue

```
mariadb> ALTER TABLE logi ADD PARTITION (PARTITION mniejsze
_od_2025 VALUES LESS THAN (2025));
1481 -
MAXVALUE can only be used in last partition definition
mariadb>
```

# Przykład użycia partycjonowania

```
CREATE PROCEDURE `generuj_zakupy`(IN `vile` int, in vod int, in vdo int)
BEGIN
```

```
CREATE or replace TABLE `historia_zakupow` (
  `ID_historia_zakupow` int(11) unsigned NOT NULL AUTO_INCREMENT,
  `ID_produkct` int(11) unsigned DEFAULT NULL,
  `ID_user` int(11) unsigned DEFAULT NULL,
  `ts_hz` datetime NULL DEFAULT NULL,
  PRIMARY KEY (`ID_historia_zakupow`)
) ENGINE=InnoDB ;
```

Tabela bez partycji

```
start TRANSACTION;
```

```
for k in 1..vile do
```

```
insert into historia_zakupow (ID_produkct, ID_user, ts_hz) values (rand() * 10+1,rand() * 10+1,from_
unixtime( rand() * (vod - vdo) + vdo));
```

```
end for;
```

```
commit;
```

```
END
```

Funkcja generująca tabelę z losowymi ID PK i FK oraz datami na podstawie przedziału dat wpisanego przez użytkownika



# Przykład użycia partycjonowania

```
mariadb> call generuj_zakupy(500000, unix_timestamp('2018-01-01 00:00:00'), unix_timestamp('2021-05-01 00:00:00'));
```

Query OK, 500000 rows affected (6.74 sec)

```
mariadb> select count(*) from historia_zakupow where year(ts_hz) = 2019;
```

count(*)
150209

1 row in set (0.17 sec)

```
mariadb> select count(*) from historia_zakupow where year(ts_hz) = 2020;
```

count(*)
150451

1 row in set (0.18 sec)

```
mariadb> select count(*) from historia_zakupow where year(ts_hz) = 2021;
```

count(*)
49125

1 row in set (0.18 sec)

Badamy właściwą liczbę rekordów do testów

# Przykład użycia partycjonowania

```
mariadb> call generuj_zakupy(1000000, unix_timestamp('2018-01-01 00:00:00'), unix_timestamp('2021-05-01 00:00:00'));
```

Query OK, 1000000 rows affected (13.47 sec)

```
mariadb> select count(*) from historia_zakupow where year(ts_hz) = 2018;
```

```
+-----+  
| count(*) |  
+-----+  
|   300233 |  
+-----+
```

1 row in set (0.35 sec)

```
mariadb> select count(*) from historia_zakupow where year(ts_hz) = 2019;
```

```
+-----+  
| count(*) |  
+-----+  
|   299946 |  
+-----+
```

1 row in set (0.35 sec)

```
mariadb> select count(*) from historia_zakupow where year(ts_hz) = 2020;
```

```
+-----+  
| count(*) |  
+-----+  
|   300938 |  
+-----+
```

1 row in set (0.35 sec)

**Badamy właściwą liczbę  
rekordów do testów**

# Przykład użycia partycjonowania

```
mariadb> call generuj_zakupy(3000000, unix_timestamp('2018-01-01 00:00:00'), unix_timestamp('2021-05-01 00:00:00'));
```

Query OK, 3000000 rows affected (41.13 sec)

```
mariadb> select count(*) from historia_zakupow where year(ts_hz) = 2020;
```

count(*)
903629

1 row in set (0.98 sec)

```
mariadb> select count(*) from historia_zakupow where year(ts_hz) = 2019;
```

count(*)
900661

1 row in set (0.99 sec)

```
mariadb> select count(*) from historia_zakupow where year(ts_hz) = 2015;
```

count(*)
0

1 row in set (0.94 sec)

Badamy właściwą liczbę rekordów do testów

# Przykład użycia partycjonowania

```
mariadb> call generuj_zakupy(6000000, unix_timestamp('2018-01-01 00:00:00'), unix_timestamp('2021-05-01 00:00:00'));
```

Query OK, 6000000 rows affected (81.29 sec)

```
mariadb> select count(*) from historia_zakupow where year(ts_hz) = 2015;
```

count(*)
0

1 row in set (1.88 sec)

```
mariadb> select count(*) from historia_zakupow where year(ts_hz) = 2018;
```

count(*)
1802049

1 row in set (1.94 sec)

```
mariadb> select count(*) from historia_zakupow where year(ts_hz) = 2019;
```

count(*)
1801544

1 row in set (1.92 sec)

Badamy właściwą liczbę rekordów do testów

# Przykład użycia partycjonowania

```
mariadb> call generuj_zakupy(10000000, unix_timestamp('2018-01-01 00:00:00'), unix_timestamp('2021-05-01 00:00:00'));
```

Query OK, 10000000 rows affected (144.75 sec)

```
mariadb> select count(*) from historia_zakupow where year(ts_hz)=2020;
```

```
+-----+  
| count(*) |  
+-----+  
| 3007376 |  
+-----+
```

1 row in set (4.03 sec)

```
mariadb> select count(*) from historia_zakupow where year(ts_hz)=2019;
```

```
+-----+  
| count(*) |  
+-----+  
| 3004079 |  
+-----+
```

1 row in set (3.83 sec)

```
mariadb> select count(*) from historia_zakupow where year(ts_hz)=2012;
```

```
+-----+  
| count(*) |  
+-----+  
| 0 |  
+-----+
```

1 row in set (3.69 sec)

```
mariadb>
```

Podstawy baz danych / Artur Niewiarowski

Dziesięć milionów rekordów będzie nadawało się do testów

# Przykład użycia partycjonowania

```
Create table historia_zakupow_part select * from historia_zakupow;
```

```
mariadb> ALTER TABLE `historia_zakupow_part`  
PARTITION BY RANGE (year(ts_hz))  
PARTITIONS 5  
(PARTITION `p_do_2018` VALUES LESS THAN (2019) MAX_ROWS = 0 MIN_ROWS = 0 ,  
PARTITION `p_2019` VALUES LESS THAN (2020) MAX_ROWS = 0 MIN_ROWS = 0 ,  
PARTITION `p_2020` VALUES LESS THAN (2021) MAX_ROWS = 0 MIN_ROWS = 0 ,  
PARTITION `p_2021` VALUES LESS THAN (2022) MAX_ROWS = 0 MIN_ROWS = 0 ,  
PARTITION `p_max` VALUES LESS THAN (maxvalue) MAX_ROWS = 0 MIN_ROWS = 0 )  
;  
Query OK, 10000000 rows affected (72.25 sec)  
Records: 10000000 Duplicates: 0 Warnings: 0
```

Partycjonujemy tabelę

# Przykład użycia partycjonowania

historia_zakupow.frm	03.05.2021 13:09	Plik FRM	2 KB
historia_zakupow.ibd	03.05.2021 14:43	Plik IBD	389 120 KB
historia_zakupow_part#p#p_2019.ibd	03.05.2021 14:02	Plik IBD	118 784 KB
historia_zakupow_part#p#p_2020.ibd	03.05.2021 14:02	Plik IBD	118 784 KB
historia_zakupow_part#p#p_2021.ibd	03.05.2021 14:02	Plik IBD	40 960 KB
historia_zakupow_part#p#p_do_2018.ibd	03.05.2021 14:02	Plik IBD	118 784 KB
historia_zakupow_part#p#p_max.ibd	03.05.2021 14:01	Plik IBD	96 KB
historia_zakupow_part.frm	03.05.2021 14:01	Plik FRM	2 KB
historia_zakupow_part.par	03.05.2021 14:01	Plik PAR	1 KB

Dziesięć milionów rekordów

# Przykład użycia partycjonowania

```
CREATE TABLE `historia_zakupow_part` (`ID_historia_zakupow` int(11) unsigned NOT NULL AUTO_INCREMENT, `ID_produkt` int(11) unsigned DEFAULT NULL, `ID_user` int(11) unsigned DEFAULT NULL, `ts_hz` datetime NOT NULL DEFAULT current_timestamp() ON UPDATE current_timestamp(), PRIMARY KEY (`ID_historia_zakupow`,`ts_hz`) USING BTREE) ENGINE=InnoDB AUTO_INCREMENT=1000001 DEFAULT CHARSET=utf8 PARTITION BY RANGE (year(`ts_hz`))(PARTITION `p_do_2018` VALUES LESS THAN (2019) ENGINE = InnoDB, PARTITION `p_2019` VALUES LESS THAN (2020) ENGINE = InnoDB, PARTITION `p_2020` VALUES LESS THAN (2021) ENGINE = InnoDB, PARTITION `p_2021` VALUES LESS THAN (2022) ENGINE = InnoDB, PARTITION `p_max` VALUES LESS THAN MAXVALUE ENGINE = InnoDB);
```

SQL DDL tabeli po partycjonowaniu



# Przykład użycia partycjonowania

```
mariadb> select count(*) from historia_zakupow_part partition(p_do_2018);
+-----+
| count(*) |
+-----+
| 3001852 |
+-----+
1 row in set (1.07 sec)
mariadb> select count(*) from historia_zakupow_part partition(p_2019);
+-----+
| count(*) |
+-----+
| 3004079 |
+-----+
1 row in set (0.67 sec)
mariadb> select count(*) from historia_zakupow_part partition(p_2020);
+-----+
| count(*) |
+-----+
| 3007376 |
+-----+
1 row in set (0.60 sec)
mariadb> select count(*) from historia_zakupow_part partition(p_2021);
+-----+
| count(*) |
+-----+
| 986693 |
+-----+
1 row in set (0.23 sec)
mariadb> select count(*) from historia_zakupow_part partition(p_max);
+-----+
| count(*) |
+-----+
| 0 |
+-----+
1 row in set (0.02 sec)
mariadb> select count(*) from historia_zakupow_part;
+-----+
| count(*) |
+-----+
| 10000000 |
+-----+
1 row in set (2.44 sec)
```

Liczba rekordów w poszczególnych partycjach

# Przykład użycia partycjonowania

```
mariadb> select count(*) from historia_zakupow  
where ts_hz = '2030-01-01';
```

```
+-----+  
| count(*) |  
+-----+  
|          0 |  
+-----+
```

```
1 row in set (3.95 sec)
```

```
mariadb> select count(*) from historia_zakupow_part  
where ts_hz = '2030-01-01';
```

```
+-----+  
| count(*) |  
+-----+  
|          0 |  
+-----+
```

```
1 row in set (0.03 sec)
```

**Test1 - brak rekordów o tej dacie, w tabeli \_part  
partycja dla tej daty jest pusta**

# Przykład użycia partycjonowania

```
mariadb> select count(*) from historia_zakupow  
where ts_hz = '2020-01-01';
```

```
+-----+  
| count(*) |  
+-----+  
|         0 |  
+-----+
```

1 row in set (4.11 sec)

```
mariadb> select count(*) from historia_zakupow_part  
where ts_hz = '2020-01-01';
```

```
+-----+  
| count(*) |  
+-----+  
|         0 |  
+-----+
```

1 row in set (0.77 sec)

# Przykład użycia partycjonowania

```
mariadb> select count(*) from historia_zakupow  
where ts_hz between '2019-01-01' and '2019-01-31';
```

```
+-----+  
| count(*) |  
+-----+  
|  246995 |  
+-----+  
1 row in set (3.94 sec)
```

```
mariadb> select count(*) from historia_zakupow_part  
where ts_hz between '2019-01-01' and '2019-01-31';
```

```
+-----+  
| count(*) |  
+-----+  
|  246995 |  
+-----+  
1 row in set (0.68 sec)
```

**Test3 - w tabeli \_part analizowana jest jedna partycja**

# Przykład użycia partycjonowania

```
mariadb> select count(*) from historia_zakupow
where ts_hz between '2019-05-01' and '2020-05-01';
```

```
+-----+
| count(*) |
+-----+
| 3011050 |
+-----+
```

```
1 row in set (3.96 sec)
```

```
mariadb> select count(*) from historia_zakupow_part
where ts_hz between '2019-05-01' and '2020-05-01';
```

```
+-----+
| count(*) |
+-----+
| 3011050 |
+-----+
```

```
1 row in set (1.46 sec)
```

```
mariadb> explain partitions select count(*) from historia_zakupow_part
where ts_hz between '2019-05-01' and '2020-05-01';
```

```
+-----+-----+-----+-----+-----+-----+-----+-----+
| id | select_type | table | partitions | type | possible_keys | key |
key_len | ref | rows | Extra |
+-----+-----+-----+-----+-----+-----+-----+-----+
| 1 | SIMPLE | historia_zakupow_part | p_2019,p_2020 | index | NULL | PRIMARY |
9 | NULL | 5997888 | Using where; Using index |
```

```
Podstawy baz danych / Artur Niewiarowski
```

```
1 row in set (0.02 sec)
```

Test4 - w tabeli \_part analizowane są dwie partycje

# Przykład użycia partycjonowania

