Architecture of Enterprise Applications 14
MySQL Partitioning

Haopeng Chen

***RE**liable, **IN**telligent and **S**calable Systems Group (**REINS**)*
Shanghai Jiao Tong University
Shanghai, China
http://reins.se.sjtu.edu.cn/~chenhp
e-mail: chen-hp@sjtu.edu.cn

Contents and Objectives

- Contents
  - Partitioning
  - From
    - https://dev.mysql.com/doc/refman/8.0/en/partitioning.html

- Objectives
  - 能够根据数据访问的具体场景，设计数据库分区方案，并能够对分区数据进行有效管理和控制

Overview of Partitioning in MySQL

- In MySQL 8.0,
  - partitioning support is provided by the InnoDB and NDB storage engines.

- The SQL standard does not provide much in the way of guidance regarding the physical aspects of data storage.
  - Nonetheless, most advanced database management systems have evolved some means of determining the physical location to be used for storing specific pieces of data in terms of the file system, hardware or even both.

- In MySQL,
  - the InnoDB storage engine has long supported the notion of a tablespace, and the MySQL Server, even prior to the introduction of partitioning, could be configured to employ different physical directories for storing different databases.

Overview of Partitioning in MySQL

- Partitioning takes this notion a step further,
  - by enabling you to distribute portions of individual tables across a file system according to rules which you can set largely as needed.
  - In effect, different portions of a table are stored as separate tables in different locations.
  - The user-selected rule by which the division of data is accomplished is known as a partitioning function, which in MySQL can be the modulus, simple matching against a set of ranges or value lists, an internal hashing function, or a linear hashing function.
  - The function is selected according to the partitioning type specified by the user, and takes as its parameter the value of a user-supplied expression.
  - This expression can be a column value, a function acting on one or more column values, or a set of one or more column values, depending on the type of partitioning that is used.

Overview of Partitioning in MySQL

- This is known as horizontal partitioning
  - —that is, different rows of a table may be assigned to different physical partitions.
  - MySQL 8.0 does not support vertical partitioning, in which different columns of a table are assigned to different physical partitions.
  - There are no plans at this time to introduce vertical partitioning into MySQL.

Overview of Partitioning in MySQL

- Some advantages of partitioning are listed here:
  - Partitioning makes it possible to store more data in one table than can be held on a single disk or file system partition.
  - Data that loses its usefulness can often be easily removed from a partitioned table by dropping the partition (or partitions) containing only that data. Conversely, the process of adding new data can in some cases be greatly facilitated by adding one or more new partitions for storing specifically that data.
  - Some queries can be greatly optimized in virtue of the fact that data satisfying a given WHERE clause can be stored only on one or more partitions, which automatically excludes any remaining partitions from the search.
  - In addition, MySQL supports explicit partition selection for queries. For example, SELECT * FROM t PARTITION (p0,p1) WHERE c < 5 selects only those rows in partitions p0 and p1 that match the WHERE condition.
    - In this case, MySQL does not check any other partitions of table t; this can greatly speed up queries when you already know which partition or partitions you wish to examine.
    - Partition selection is also supported for the data modification statements DELETE, INSERT, REPLACE, UPDATE, and LOAD DATA, LOAD XML.

Partitioning Types

- The types of partitioning which are available in MySQL 8.0 are listed here:
  - **RANGE partitioning.** This type of partitioning assigns rows to partitions based on column values falling within a given range.
  - **LIST partitioning.** Similar to partitioning by RANGE, except that the partition is selected based on columns matching one of a set of discrete values.
  - **HASH partitioning.** With this type of partitioning, a partition is selected based on the value returned by a user-defined expression that operates on column values in rows to be inserted into the table. The function may consist of any expression valid in MySQL that yields a nonnegative integer value.
  - **KEY partitioning.** This type of partitioning is similar to partitioning by HASH, except that only one or more columns to be evaluated are supplied, and the MySQL server provides its own hashing function. These columns can contain other than integer values, since the hashing function supplied by MySQL guarantees an integer result regardless of the column data type.

Partitioning Types

- A very common use of database partitioning is to segregate data by date.

```
CREATE TABLE members (
        firstname VARCHAR(25) NOT NULL, lastname VARCHAR(25) NOT NULL,
        username VARCHAR(16) NOT NULL, email VARCHAR(35),
        joined DATE NOT NULL
)
PARTITION BY KEY(joined)
PARTITIONS 6;
```

- Other partitioning types require a partitioning expression that yields an integer value or NULL.

```
CREATE TABLE members (
        firstname VARCHAR(25) NOT NULL, lastname VARCHAR(25) NOT NULL,
        username VARCHAR(16) NOT NULL, email VARCHAR(35),
        joined DATE NOT NULL
)
PARTITION BY RANGE( YEAR(joined) ) (
        PARTITION p0 VALUES LESS THAN (1960), PARTITION p1 VALUES LESS THAN (1970),
        PARTITION p2 VALUES LESS THAN (1980), PARTITION p3 VALUES LESS THAN (1990),
        PARTITION p4 VALUES LESS THAN MAXVALUE
);
```

RANGE Partitioning

- A table that is partitioned by range is partitioned in such a way
  - that each partition contains rows for which the partitioning expression value lies within a given range.
  - Ranges should be contiguous but not overlapping, and are defined using the VALUES LESS THAN operator.

```
CREATE TABLE employees (
  id INT NOT NULL,
  fname VARCHAR(30),
  lname VARCHAR(30),
  hired DATE NOT NULL DEFAULT '1970-01-01',
  separated DATE NOT NULL DEFAULT '9999-12-31',
  job_code INT NOT NULL,
  store_id INT NOT NULL
);
```

RANGE Partitioning

- This table can be partitioned by range in a number of ways, depending on your needs.

```
CREATE TABLE employees (
        id INT NOT NULL,
        fname VARCHAR(30),
        lname VARCHAR(30),
        hired DATE NOT NULL DEFAULT '1970-01-01',
        separated DATE NOT NULL DEFAULT '9999-12-31',
        job_code INT NOT NULL, store_id INT NOT NULL
)
PARTITION BY RANGE (store_id) (
        PARTITION p0 VALUES LESS THAN (6),
        PARTITION p1 VALUES LESS THAN (11),
        PARTITION p2 VALUES LESS THAN (16),
        PARTITION p3 VALUES LESS THAN (21)
);
```

RANGE Partitioning

- This table can be partitioned by range in a number of ways, depending on your needs.

```
CREATE TABLE employees (
        id INT NOT NULL,
        fname VARCHAR(30),
        lname VARCHAR(30),
        hired DATE NOT NULL DEFAULT '1970-01-01',
        separated DATE NOT NULL DEFAULT '9999-12-31',
        job_code INT NOT NULL, store_id INT NOT NULL
)
PARTITION BY RANGE (store_id) (
        PARTITION p0 VALUES LESS THAN (6),
        PARTITION p1 VALUES LESS THAN (11),
        PARTITION p2 VALUES LESS THAN (16),
        PARTITION p3 VALUES LESS THAN MAXVALUE
);
```

RANGE Partitioning

- This table can be partitioned by range in a number of ways, depending on your needs.

```
CREATE TABLE employees (
        id INT NOT NULL,
        fname VARCHAR(30),
        lname VARCHAR(30),
        hired DATE NOT NULL DEFAULT '1970-01-01',
        separated DATE NOT NULL DEFAULT '9999-12-31',
        job_code INT NOT NULL, store_id INT NOT NULL
)
PARTITION BY RANGE (job_code) (
        PARTITION p0 VALUES LESS THAN (100),
        PARTITION p1 VALUES LESS THAN (1000),
        PARTITION p2 VALUES LESS THAN (10000)
);
```

RANGE Partitioning

- This table can be partitioned by range in a number of ways, depending on your needs.

```
CREATE TABLE employees (
        id INT NOT NULL,
        fname VARCHAR(30),
        lname VARCHAR(30),
        hired DATE NOT NULL DEFAULT '1970-01-01',
        separated DATE NOT NULL DEFAULT '9999-12-31',
        job_code INT NOT NULL, store_id INT NOT NULL
)
PARTITION BY RANGE ( YEAR(separated) ) (
        PARTITION p0 VALUES LESS THAN (1991),
        PARTITION p1 VALUES LESS THAN (1996),
        PARTITION p2 VALUES LESS THAN (2001),
        PARTITION p3 VALUES LESS THAN MAXVALUE
);
```

RANGE Partitioning

- This table can be partitioned by range in a number of ways, depending on your needs.

```
CREATE TABLE quarterly_report_status (
        report_id INT NOT NULL,
        report_status VARCHAR(20) NOT NULL,
        report_updated TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP
)
PARTITION BY RANGE ( UNIX_TIMESTAMP(report_updated) ) (
        PARTITION p0 VALUES LESS THAN ( UNIX_TIMESTAMP('2008-01-01 00:00:00') ),
        PARTITION p1 VALUES LESS THAN ( UNIX_TIMESTAMP('2008-04-01 00:00:00') ),
        PARTITION p2 VALUES LESS THAN ( UNIX_TIMESTAMP('2008-07-01 00:00:00') ),
        PARTITION p3 VALUES LESS THAN ( UNIX_TIMESTAMP('2008-10-01 00:00:00') ),
        PARTITION p4 VALUES LESS THAN ( UNIX_TIMESTAMP('2009-01-01 00:00:00') ),
        PARTITION p5 VALUES LESS THAN ( UNIX_TIMESTAMP('2009-04-01 00:00:00') ),
        PARTITION p6 VALUES LESS THAN ( UNIX_TIMESTAMP('2009-07-01 00:00:00') ),
        PARTITION p7 VALUES LESS THAN ( UNIX_TIMESTAMP('2009-10-01 00:00:00') ),
        PARTITION p8 VALUES LESS THAN ( UNIX_TIMESTAMP('2010-01-01 00:00:00') ),
        PARTITION p9 VALUES LESS THAN (MAXVALUE)
);
```

RANGE Partitioning

- Partition the table by RANGE,

```
CREATE TABLE members (
        firstname VARCHAR(25) NOT NULL,
        lastname VARCHAR(25) NOT NULL,
        username VARCHAR(16) NOT NULL,
        email VARCHAR(35),
        joined DATE NOT NULL
)
PARTITION BY RANGE( YEAR(joined) ) (
        PARTITION p0 VALUES LESS THAN (1960),
        PARTITION p1 VALUES LESS THAN (1970),
        PARTITION p2 VALUES LESS THAN (1980),
        PARTITION p3 VALUES LESS THAN (1990),
        PARTITION p4 VALUES LESS THAN MAXVALUE
);
```

- Partition the table by RANGE COLUMNS

```
PARTITION BY RANGE COLUMNS(joined) (
        PARTITION p0 VALUES LESS THAN ('1960-01-01'),
        PARTITION p1 VALUES LESS THAN ('1970-01-01'),
        PARTITION p2 VALUES LESS THAN ('1980-01-01'),
        PARTITION p3 VALUES LESS THAN ('1990-01-01'),
        PARTITION p4 VALUES LESS THAN MAXVALUE
);
```

LIST Partitioning

- List partitioning in MySQL is similar to range partitioning in many ways.
  - The chief difference between the two types of partitioning is that, in list partitioning, each partition is defined and selected based on the membership of a column value in one of a set of value lists.

```
CREATE TABLE employees (
    id INT NOT NULL,
    fname VARCHAR(30),
    lname VARCHAR(30),
    hired DATE NOT NULL DEFAULT '1970-01-01',
    separated DATE NOT NULL DEFAULT '9999-12-31',
    job_code INT,
    store_id INT
);
```

LIST Partitioning

- Suppose that there are 20 video stores distributed among 4 franchises as shown in the following table.

Region	Store ID Numbers
North	3, 5, 6, 9, 17
East	1, 2, 10, 11, 19, 20
West	4, 12, 13, 14, 18
Central	7, 8, 15, 16

LIST Partitioning

- List partitioning in MySQL is similar to range partitioning in many ways.

```
CREATE TABLE employees (
        id INT NOT NULL,
        fname VARCHAR(30),
        lname VARCHAR(30),
        hired DATE NOT NULL DEFAULT '1970-01-01',
        separated DATE NOT NULL DEFAULT '9999-12-31',
        job_code INT,
        store_id INT
)
PARTITION BY LIST(store_id) (
        PARTITION pNorth VALUES IN (3,5,6,9,17),
        PARTITION pEast VALUES IN (1,2,10,11,19,20),
        PARTITION pWest VALUES IN (4,12,13,14,18),
        PARTITION pCentral VALUES IN (7,8,15,16)
);
```

LIST Partitioning

- Unlike the case with RANGE partitioning, there is no “catch-all” such as MAXVALUE;
  - all expected values for the partitioning expression should be covered in PARTITION ... VALUES IN (...) clauses.

```
mysql> CREATE TABLE h2 (
    -> c1 INT,
    -> c2 INT
    -> )
    -> PARTITION BY LIST(c1) (
    -> PARTITION p0 VALUES IN (1, 4, 7),
    -> PARTITION p1 VALUES IN (2, 5, 8)
    -> );
Query OK, 0 rows affected (0.11 sec)
mysql> INSERT INTO h2 VALUES (3, 5);
ERROR 1525 (HY000): Table has no partition for value 3
```

LIST Partitioning

- You can cause this type of error to be ignored by using the IGNORE keyword.

```
mysql> TRUNCATE h2;
Query OK, 1 row affected (0.00 sec)
mysql> SELECT * FROM h2;
Empty set (0.00 sec)
mysql> INSERT IGNORE INTO h2 VALUES (2, 5), (6, 10), (7, 5), (3, 1), (1, 9);
Query OK, 3 rows affected (0.00 sec)
Records: 5 Duplicates: 2 Warnings: 0
mysql> SELECT * FROM h2;
+------+------+
| c1   | c2   |
+------+------+
| 7    | 5    |
| 1    | 9    |
| 2    | 5    |
+------+------+
3 rows in set (0.00 sec)
```

COLUMN Partitioning

- COLUMNS partitioning is variants on RANGE and LIST partitioning.
  - COLUMNS partitioning enables the use of multiple columns in partitioning keys.
  - All of these columns are taken into account both for the purpose of placing rows in partitions and for the determination of which partitions are to be checked for matching rows in partition pruning.

- Both RANGE COLUMNS partitioning and LIST COLUMNS partitioning support the use of non-integer columns for defining value ranges or list members.
  - All integer types: TINYINT, SMALLINT, MEDIUMINT, INT (INTEGER), and BIGINT. (This is the same as with partitioning by RANGE and LIST.)

    Other numeric data types (such as DECIMAL or FLOAT) are not supported as partitioning columns.
  - DATE and DATETIME.

    Columns using other data types relating to dates or times are not supported as partitioning columns.
  - The following string types: CHAR, VARCHAR, BINARY, and VARBINARY.

    TEXT and BLOB columns are not supported as partitioning columns.

RANGE COLUMN Partitioning

- Range columns partitioning
  - is similar to range partitioning, but enables you to define partitions using ranges based on multiple column values.

- RANGE COLUMNS partitioning differs significantly from RANGE partitioning in the following ways:
  - RANGE COLUMNS does not accept expressions, only names of columns.
  - RANGE COLUMNS accepts a list of one or more columns.
  - RANGE COLUMNS partitions are based on comparisons between tuples (lists of column values) rather than comparisons between scalar values.
  - RANGE COLUMNS partitioning columns are not restricted to integer columns; string, DATE and DATETIME columns can also be used as partitioning columns.

RANGE COLUMN Partitioning

- The basic syntax for creating a table partitioned by RANGE COLUMNS is shown here:

```
CREATE TABLE table_name
PARTITIONED BY RANGE COLUMNS(column_list) (
    PARTITION partition_name VALUES LESS THAN (value_list)[,
    PARTITION partition_name VALUES LESS THAN (value_list)][,
    ...] )
column_list:
    column_name[, column_name][, ...]
value_list:
    value[, value][, ...]
```

RANGE COLUMN Partitioning


```
mysql> CREATE TABLE rcx (
    -> a INT,
    -> b INT,
    -> c CHAR(3),
    -> d INT
    -> )
    -> PARTITION BY RANGE COLUMNS(a,d,c) (
    -> PARTITION p0 VALUES LESS THAN (5,10,'ggg'),
    -> PARTITION p1 VALUES LESS THAN (10,20,'mmm'),
    -> PARTITION p2 VALUES LESS THAN (15,30,'sss'),
    -> PARTITION p3 VALUES LESS THAN (MAXVALUE,MAXVALUE,MAXVALUE)
    -> );
Query OK, 0 rows affected (0.15 sec)
```

RANGE COLUMN Partitioning


```
CREATE TABLE rc1 ( a INT, b INT )
PARTITION BY RANGE COLUMNS(a, b) (
        PARTITION p0 VALUES LESS THAN (5, 12),
        PARTITION p3 VALUES LESS THAN (MAXVALUE, MAXVALUE)
);

mysql> INSERT INTO rc1 VALUES (5,10), (5,11), (5,12);
Query OK, 3 rows affected (0.00 sec)
Records: 3 Duplicates: 0 Warnings: 0

mysql> SELECT PARTITION_NAME,TABLE_ROWS
    -> FROM INFORMATION_SCHEMA.PARTITIONS
    -> WHERE TABLE_NAME = 'rc1';
+--------------------+------------------------+------------------+
| TABLE_SCHEMA | PARTITION_NAME | TABLE_ROWS |
+--------------------+------------------------+------------------+
| p                  | p0                     | 2                |
| p                  | p1                     | 1                |
+--------------------+------------------------+------------------+
2 rows in set (0.00 sec)
```

RANGE COLUMN Partitioning


```
CREATE TABLE rx ( a INT, b INT )
PARTITION BY RANGE COLUMNS (a) (
        PARTITION p0 VALUES LESS THAN (5),
        PARTITION p1 VALUES LESS THAN (MAXVALUE)
);

mysql> INSERT INTO rx VALUES (5,10), (5,11), (5,12);
Query OK, 3 rows affected (0.00 sec)
Records: 3 Duplicates: 0 Warnings: 0

mysql> SELECT PARTITION_NAME,TABLE_ROWS
    -> FROM INFORMATION_SCHEMA.PARTITIONS
    -> WHERE TABLE_NAME = 'rx';
+--------------+----------------+------------+
| TABLE_SCHEMA | PARTITION_NAME | TABLE_ROWS |
+--------------+----------------+------------+
| p            | p0             | 0          |
| p            | p1             | 3          |
+--------------+----------------+------------+
2 rows in set (0.00 sec)
```

RANGE COLUMN Partitioning


```
CREATE TABLE rc4 ( a INT, b INT, c INT )
PARTITION BY RANGE COLUMNS(a,b,c) (
    PARTITION p0 VALUES LESS THAN (0,25,50),
    PARTITION p1 VALUES LESS THAN (10,20,100),
    PARTITION p2 VALUES LESS THAN (10,30,50)
    PARTITION p3 VALUES LESS THAN (MAXVALUE,MAXVALUE,MAXVALUE)
);

mysql> SELECT (0,25,50) < (10,20,100), (10,20,100) < (10,30,50);
+--------------------------+-------------------------------+
| (0,25,50) < (10,20,100) | (10,20,100) < (10,30,50) |
+--------------------------+-------------------------------+
|                        1 |                             1 |
+--------------------------+-------------------------------+
1 row in set (0.00 sec)
```

RANGE COLUMN Partitioning


```
mysql> CREATE TABLE rcf (
    -> a INT,
    -> b INT,
    -> c INT
    -> )
    -> PARTITION BY RANGE COLUMNS(a,b,c) (
    -> PARTITION p0 VALUES LESS THAN (0,25,50),
    -> PARTITION p1 VALUES LESS THAN (20,20,100),
    -> PARTITION p2 VALUES LESS THAN (10,30,50),
    -> PARTITION p3 VALUES LESS THAN (MAXVALUE,MAXVALUE,MAXVALUE)
    -> );
ERROR 1493 (HY000): VALUES LESS THAN value must be strictly increasing for each partition

mysql> SELECT (0,25,50) < (20,20,100), (20,20,100) < (10,30,50);
+-----------------------------+-----------------------------+
| (0,25,50) < (20,20,100) | (20,20,100) < (10,30,50)  |
+-----------------------------+-----------------------------+
|                           1 |                           0 |
+-----------------------------+-----------------------------+
1 row in set (0.00 sec)
```

RANGE COLUMN Partitioning


```
CREATE TABLE employees_by_lname (
    id INT NOT NULL,
    fname VARCHAR(30),
    lname VARCHAR(30),
    hired DATE NOT NULL DEFAULT '1970-01-01',
    separated DATE NOT NULL DEFAULT '9999-12-31',
    job_code INT NOT NULL,
    store_id INT NOT NULL
)
PARTITION BY RANGE COLUMNS (lname) (
    PARTITION p0 VALUES LESS THAN ('g'),
    PARTITION p1 VALUES LESS THAN ('m'),
    PARTITION p2 VALUES LESS THAN ('t'),
    PARTITION p3 VALUES LESS THAN (MAXVALUE)
);
```

RANGE COLUMN Partitioning


```
ALTER TABLE employees PARTITION BY RANGE COLUMNS (lname) (
    PARTITION p0 VALUES LESS THAN ('g'),
    PARTITION p1 VALUES LESS THAN ('m'),
    PARTITION p2 VALUES LESS THAN ('t'),
    PARTITION p3 VALUES LESS THAN (MAXVALUE)
);

ALTER TABLE employees PARTITION BY RANGE COLUMNS (hired) (
    PARTITION p0 VALUES LESS THAN ('1970-01-01'),
    PARTITION p1 VALUES LESS THAN ('1980-01-01'),
    PARTITION p2 VALUES LESS THAN ('1990-01-01'),
    PARTITION p3 VALUES LESS THAN ('2000-01-01'),
    PARTITION p4 VALUES LESS THAN ('2010-01-01'),
    PARTITION p5 VALUES LESS THAN (MAXVALUE)
);
```

LIST COLUMN Partitioning

- MySQL 8.0 provides support for LIST COLUMNS partitioning.
  - This is a variant of LIST partitioning that enables the use of multiple columns as partition keys
- Suppose that you have a business that has customers in 12 cities which,
  - for sales and marketing purposes, you organize into 4 regions of 3 cities each as shown in the following table:

Region	Cities
1	Oskarshamn, Högsby, Mönsterås
2	Vimmerby, Hultsfred, Västervik
3	Nässjö, Eksjö, Vetlanda
4	Uppvidinge, Alvesta, Växjo

LIST COLUMN Partitioning


```
CREATE TABLE customers_1 (
    first_name VARCHAR(25),
    last_name VARCHAR(25),
    street_1 VARCHAR(30),
    street_2 VARCHAR(30),
    city VARCHAR(15),
    renewal DATE
)
PARTITION BY LIST COLUMNS(city) (
    PARTITION pRegion_1 VALUES IN('Oskarshamn', 'Högsby', 'Mönsterås'),
    PARTITION pRegion_2 VALUES IN('Vimmerby', 'Hultsfred', 'Västervik'),
    PARTITION pRegion_3 VALUES IN('Nässjö', 'Eksjö', 'Vetlanda'),
    PARTITION pRegion_4 VALUES IN('Uppvidinge', 'Alvesta', 'Växjo')
);
```

LIST COLUMN Partitioning


```
CREATE TABLE customers_1 (
        first_name VARCHAR(25),
        last_name VARCHAR(25),
        street_1 VARCHAR(30),
        street_2 VARCHAR(30),
        city VARCHAR(15),
        renewal DATE
)
PARTITION BY LIST COLUMNS(renewal) (
        PARTITION pWeek_1 VALUES IN('2010-02-01', '2010-02-02', '2010-02-03',
                '2010-02-04', '2010-02-05', '2010-02-06', '2010-02-07'),
        PARTITION pWeek_2 VALUES IN('2010-02-08', '2010-02-09', '2010-02-10',
                '2010-02-11', '2010-02-12', '2010-02-13', '2010-02-14'),
        PARTITION pWeek_3 VALUES IN('2010-02-15', '2010-02-16', '2010-02-17',
                '2010-02-18', '2010-02-19', '2010-02-20', '2010-02-21'),
        PARTITION pWeek_4 VALUES IN('2010-02-22', '2010-02-23', '2010-02-24',
                '2010-02-25', '2010-02-26', '2010-02-27', '2010-02-28')
);
```

LIST COLUMN Partitioning


```
CREATE TABLE customers_1 (
    first_name VARCHAR(25),
    last_name VARCHAR(25),
    street_1 VARCHAR(30),
    street_2 VARCHAR(30),
    city VARCHAR(15),
    renewal DATE
)
PARTITION BY RANGE COLUMNS(renewal) (
    PARTITION pWeek_1 VALUES LESS THAN('2010-02-09'),
    PARTITION pWeek_2 VALUES LESS THAN('2010-02-15'),
    PARTITION pWeek_3 VALUES LESS THAN('2010-02-22'),
    PARTITION pWeek_4 VALUES LESS THAN('2010-03-01')
);
```

HASH Partitioning

- Partitioning by HASH is used primarily
  - to ensure an even distribution of data among a predetermined number of partitions.

```
CREATE TABLE employees (
    id INT NOT NULL,
    fname VARCHAR(30),
    lname VARCHAR(30),
    hired DATE NOT NULL DEFAULT '1970-01-01',
    separated DATE NOT NULL DEFAULT '9999-12-31',
    job_code INT, store_id INT
)
PARTITION BY HASH(store_id)
PARTITIONS 4;
```

HASH Partitioning

- Partitioning by HASH is used primarily
  - to ensure an even distribution of data among a predetermined number of partitions.

```
CREATE TABLE employees (
    id INT NOT NULL,
    fname VARCHAR(30),
    lname VARCHAR(30),
    hired DATE NOT NULL DEFAULT '1970-01-01',
    separated DATE NOT NULL DEFAULT '9999-12-31',
    job_code INT, store_id INT
)
PARTITION BY HASH(YEAR(hired))
PARTITIONS 4;
```

HASH Partitioning


```
CREATE TABLE t1 (col1 INT, col2 CHAR(5), col3 DATE)
PARTITION BY HASH( YEAR(col3) )
PARTITIONS 4;
```

```
MOD(YEAR('2005-09-01'),4)
= MOD(2005,4)
= 1
```

LINEAR HASH Partitioning

- MySQL also supports linear hashing,
  - which differs from regular hashing in that linear hashing utilizes a linear powers-of-two algorithm whereas regular hashing employs the modulus of the hashing function's value.

```
CREATE TABLE employees (
    id INT NOT NULL,
    fname VARCHAR(30),
    lname VARCHAR(30),
    hired DATE NOT NULL DEFAULT '1970-01-01',
    separated DATE NOT NULL DEFAULT '9999-12-31',
    job_code INT, store_id INT
)
PARTITION BY LINEAR HASH(YEAR(hired))
PARTITIONS 4;
```

LINEAR HASH Partitioning

- Given an expression *expr*,
  - the partition in which the record is stored when linear hashing is used is partition number *N* from among *num* partitions, where *N* is derived according to the following algorithm:

1. Find the next power of 2 greater than *num*. We call this value *V*; it can be calculated as:
   - *V* = POWER(2, CEILING(LOG(2, *num*)))
2. Set *N* = *F*(*column_list*) & (*V* - 1).
3. While *N* >= *num*:
   - Set *V* = *V* / 2
   - Set *N* = *N* & (*V* - 1)

LINEAR HASH Partitioning


```
CREATE TABLE t1 (col1 INT, col2 CHAR(5), col3 DATE)
        PARTITION BY LINEAR HASH( YEAR(col3) )
        PARTITIONS 6;
```

V = POWER(2, CEILING(LOG(2,6))) = 8
N = YEAR('2003-04-14') & (8 - 1)
= 2003 & 7
= 3
(*3 >= 6 is FALSE: record stored in partition #3*)

V = 8
N = YEAR('1998-10-19') & (8 - 1)
= 1998 & 7
= 6
(*6 >= 6 is TRUE: additional step required*)

N = 6 & ((8 / 2) - 1)
= 6 & 3
= 2
(*2 >= 6 is FALSE: record stored in partition #2*)

KEY Partitioning

- Partitioning by key is similar to partitioning by hash,
  - except that where hash partitioning employs a user-defined expression,
  - the hashing function for key partitioning is supplied by the MySQL server.

```
CREATE TABLE k1 (
      id INT NOT NULL PRIMARY KEY,
      name VARCHAR(20)
)
PARTITION BY KEY()
PARTITIONS 2;

CREATE TABLE tk ( col1 INT NOT NULL, col2 CHAR(5), col3 DATE )
PARTITION BY LINEAR KEY (col1)
PARTITIONS 3;
```

Subpartitioning

- Subpartitioning—also known as composite partitioning
  - is the further division of each partition in a partitioned table.

```
CREATE TABLE ts (id INT, purchased DATE)
    PARTITION BY RANGE( YEAR(purchased) )
    SUBPARTITION BY HASH( TO_DAYS(purchased) )
    SUBPARTITIONS 2 (
            PARTITION p0 VALUES LESS THAN (1990),
            PARTITION p1 VALUES LESS THAN (2000),
            PARTITION p2 VALUES LESS THAN MAXVALUE
    );
```

- Table ts has 3 RANGE partitions.
  - Each of these partitions—p0, p1, and p2—is further divided into 2 subpartitions.
  - In effect, the entire table is divided into 3 * 2 = 6 partitions.
  - However, due to the action of the PARTITION BY RANGE clause, the first 2 of these store only those records with a value less than 1990 in the purchased column.

Subpartitioning

- It is also possible to define subpartitions explicitly using SUBPARTITION clauses to specify options for individual subpartitions.

```
CREATE TABLE ts (id INT, purchased DATE)
PARTITION BY RANGE( YEAR(purchased) )
SUBPARTITION BY HASH( TO_DAYS(purchased) ) (
        PARTITION p0 VALUES LESS THAN (1990) (
                SUBPARTITION s0,
                SUBPARTITION s1
        ),
        PARTITION p1 VALUES LESS THAN (2000) (
                SUBPARTITION s2,
                SUBPARTITION s3
        ),
        PARTITION p2 VALUES LESS THAN MAXVALUE (
                SUBPARTITION s4,
                SUBPARTITION s5
        )
);
```

How MySQL Partitioning Handles NULL

- **Handling of NULL with RANGE partitioning**
  - If you insert a row into a table partitioned by RANGE such that the column value used to determine the partition is NULL, the row is inserted into the lowest partition.

```
mysql> CREATE TABLE t1 (
    -> c1 INT,
    -> c2 VARCHAR(20)
    -> )
    -> PARTITION BY RANGE(c1) (
    -> PARTITION p0 VALUES LESS THAN (0),
    -> PARTITION p1 VALUES LESS THAN (10),
    -> PARTITION p2 VALUES LESS THAN MAXVALUE
    -> );
Query OK, 0 rows affected (0.09 sec)
mysql> CREATE TABLE t2 (
    -> c1 INT,
    -> c2 VARCHAR(20)
    -> )
    -> PARTITION BY RANGE(c1) (
    -> PARTITION p0 VALUES LESS THAN (-5),
    -> PARTITION p1 VALUES LESS THAN (0),
    -> PARTITION p2 VALUES LESS THAN (10),
    -> PARTITION p3 VALUES LESS THAN MAXVALUE
    -> );
Query OK, 0 rows affected (0.09 sec)
```

How MySQL Partitioning Handles NULL

- **Handling of NULL with RANGE partitioning**

```
mysql> SELECT TABLE_NAME, PARTITION_NAME, TABLE_ROWS,
           AVG_ROW_LENGTH, DATA_LENGTH
    > FROM INFORMATION_SCHEMA.PARTITIONS
    > WHERE TABLE_SCHEMA = 'p' AND TABLE_NAME LIKE 't_';
+------------+----------------+------------+----------------+-------------+
| TABLE_NAME | PARTITION_NAME | TABLE_ROWS | AVG_ROW_LENGTH | DATA_LENGTH |
+------------+----------------+------------+----------------+-------------+
| t1         | p0             |          0 |              0 |           0 |
| t1         | p1             |          0 |              0 |           0 |
| t1         | p2             |          0 |              0 |           0 |
| t2         | p0             |          0 |              0 |           0 |
| t2         | p1             |          0 |              0 |           0 |
| t2         | p2             |          0 |              0 |           0 |
| t2         | p3             |          0 |              0 |           0 |
+------------+----------------+------------+----------------+-------------+
7 rows in set (0.00 sec)
```

How MySQL Partitioning Handles NULL

- **Handling of NULL with RANGE partitioning**

```
mysql> INSERT INTO t1 VALUES (NULL, 'mothra');
Query OK, 1 row affected (0.00 sec)
mysql> INSERT INTO t2 VALUES (NULL, 'mothra');
Query OK, 1 row affected (0.00 sec)
mysql> SELECT * FROM t1;
+--------+----------+
| id     | name     |
+--------+----------+
| NULL | mothra |
+--------+----------+
1 row in set (0.00 sec)
mysql> SELECT * FROM t2;
+--------+----------+
| id     | name     |
+--------+----------+
| NULL | mothra |
+--------+----------+
1 row in set (0.00 sec)
```

How MySQL Partitioning Handles NULL

- **Handling of NULL with RANGE partitioning**

```
mysql> SELECT TABLE_NAME, PARTITION_NAME, TABLE_ROWS,
           AVG_ROW_LENGTH, DATA_LENGTH
    > FROM INFORMATION_SCHEMA.PARTITIONS
    > WHERE TABLE_SCHEMA = 'p' AND TABLE_NAME LIKE 't_';
+------------+----------------+------------+----------------+-------------+
| TABLE_NAME | PARTITION_NAME | TABLE_ROWS | AVG_ROW_LENGTH | DATA_LENGTH |
+------------+----------------+------------+----------------+-------------+
| t1         | p0             |          1 |             20 |          20 |
| t1         | p1             |          0 |              0 |           0 |
| t1         | p2             |          0 |              0 |           0 |
| t2         | p0             |          1 |             20 |          20 |
| t2         | p1             |          0 |              0 |           0 |
| t2         | p2             |          0 |              0 |           0 |
| t2         | p3             |          0 |              0 |           0 |
+------------+----------------+------------+----------------+-------------+
7 rows in set (0.00 sec)
```

How MySQL Partitioning Handles NULL

- **Handling of NULL with RANGE partitioning**

```
mysql> ALTER TABLE t1 DROP PARTITION p0;
Query OK, 0 rows affected (0.16 sec)

mysql> ALTER TABLE t2 DROP PARTITION p0;
Query OK, 0 rows affected (0.16 sec)

mysql> SELECT * FROM t1;
Empty set (0.00 sec)

mysql> SELECT * FROM t2;
Empty set (0.00 sec)
```

How MySQL Partitioning Handles NULL

- **Handling of NULL with LIST partitioning**
  - A table that is partitioned by LIST admits NULL values if and only if one of its partitions is defined using that value-list that contains NULL.

```
mysql> CREATE TABLE ts1 (
    -> c1 INT,
    -> c2 VARCHAR(20)
    -> )
    -> PARTITION BY LIST(c1) (
    -> PARTITION p0 VALUES IN (0, 3, 6),
    -> PARTITION p1 VALUES IN (1, 4, 7),
    -> PARTITION p2 VALUES IN (2, 5, 8)
    -> ); Query OK, 0 rows affected (0.01 sec)
mysql> INSERT INTO ts1 VALUES (9, 'mothra');
ERROR 1504 (HY000): Table has no partition for value 9
mysql> INSERT INTO ts1 VALUES (NULL, 'mothra');
ERROR 1504 (HY000): Table has no partition for value NULL
```

How MySQL Partitioning Handles NULL

- **Handling of NULL with LIST partitioning**
  - A table that is partitioned by LIST admits NULL values if and only if one of its partitions is defined using that value-list that contains NULL.

```
mysql> CREATE TABLE ts1 (
    -> c1 INT,
    -> c2 VARCHAR(20)
    -> )
    -> PARTITION BY LIST(c1) (
    -> PARTITION p0 VALUES IN (0, 3, 6),
    -> PARTITION p1 VALUES IN (1, 4, 7),
    -> PARTITION p2 VALUES IN (2, 5, 8)
    -> ); Query OK, 0 rows affected (0.01 sec)
mysql> INSERT INTO ts1 VALUES (9, 'mothra');
ERROR 1504 (HY000): Table has no partition for value 9
mysql> INSERT INTO ts1 VALUES (NULL, 'mothra');
ERROR 1504 (HY000): Table has no partition for value NULL
```

How MySQL Partitioning Handles NULL

- **Handling of NULL with LIST partitioning**

```
mysql> CREATE TABLE ts2 (
    -> c1 INT,
    -> c2 VARCHAR(20)
    -> )
    -> PARTITION BY LIST(c1) (
    -> PARTITION p0 VALUES IN (0, 3, 6),
    -> PARTITION p1 VALUES IN (1, 4, 7),
    -> PARTITION p2 VALUES IN (2, 5, 8),
    -> PARTITION p3 VALUES IN (NULL)
    -> );
Query OK, 0 rows affected (0.01 sec)

mysql> CREATE TABLE ts3 (
    -> c1 INT,
    -> c2 VARCHAR(20)
    -> )
    -> PARTITION BY LIST(c1) (
    -> PARTITION p0 VALUES IN (0, 3, 6),
    -> PARTITION p1 VALUES IN (1, 4, 7, NULL),
    -> PARTITION p2 VALUES IN (2, 5, 8)
    -> );
Query OK, 0 rows affected (0.01 sec)
```

How MySQL Partitioning Handles NULL

- **Handling of NULL with LIST partitioning**

```
mysql> INSERT INTO ts2 VALUES (NULL, 'mothra');
Query OK, 1 row affected (0.00 sec)
mysql> INSERT INTO ts3 VALUES (NULL, 'mothra');
Query OK, 1 row affected (0.00 sec)

mysql> SELECT TABLE_NAME, PARTITION_NAME, TABLE_ROWS,
    >        AVG_ROW_LENGTH, DATA_LENGTH
    > FROM INFORMATION_SCHEMA.PARTITIONS
    > WHERE TABLE_SCHEMA = 'p' AND TABLE_NAME LIKE 't_';
+------------+----------------+------------+----------------+-------------+
| TABLE_NAME | PARTITION_NAME | TABLE_ROWS | AVG_ROW_LENGTH | DATA_LENGTH |
+------------+----------------+------------+----------------+-------------+
| ts2        | p0             |          0 |              0 |           0 |
| ts2        | p1             |          0 |              0 |           0 |
| ts2        | p2             |          0 |              0 |           0 |
| ts3        | p0             |          1 |             20 |          20 |
| ts3        | p1             |          0 |              0 |           0 |
| ts3        | p2             |          1 |             20 |          20 |
| ts3        | p3             |          0 |              0 |           0 |
+------------+----------------+------------+----------------+-------------+
7 rows in set (0.00 sec)
```

How MySQL Partitioning Handles NULL

- **Handling of NULL with HASH and KEY partitioning**
  - Any partition expression that yields a NULL value is treated as though its return value were zero.

```
mysql> CREATE TABLE th (
    -> c1 INT,
    -> c2 VARCHAR(20)
    -> )
    -> PARTITION BY HASH(c1)
    -> PARTITIONS 2;
Query OK, 0 rows affected (0.00 sec)
mysql> SELECT TABLE_NAME,PARTITION_NAME,TABLE_ROWS,AVG_ROW_LENGTH,                DATA_LENGTH
    > FROM INFORMATION_SCHEMA.PARTITIONS
    > WHERE TABLE_SCHEMA = 'p' AND TABLE_NAME ='th';
+------------+----------------+------------+-----------------+-------------+
| TABLE_NAME | PARTITION_NAME | TABLE_ROWS | AVG_ROW_LENGTH | DATA_LENGTH |
+------------+----------------+------------+-----------------+-------------+
| th         | p0             |          0 |               0 |           0 |
| th         | p1             |          0 |               0 |           0 |
+------------+----------------+------------+-----------------+-------------+
2 rows in set (0.00 sec)
```

How MySQL Partitioning Handles NULL

- **Handling of NULL with HASH and KEY partitioning**
  - Any partition expression that yields a NULL value is treated as though its return value were zero.

```
mysql> INSERT INTO th VALUES (NULL, 'mothra'), (0, 'gigan');
Query OK, 1 row affected (0.00 sec)

mysql> SELECT * FROM th;
+--------+----------+
| c1     | c2       |
+--------+----------+
| NULL   | mothra   |
+--------+----------+
| 0      | gigan    |
+--------+----------+
2 rows in set (0.01 sec)
```

How MySQL Partitioning Handles NULL

- **Handling of NULL with HASH and KEY partitioning**

```
mysql> SELECT TABLE_NAME,PARTITION_NAME,TABLE_ROWS,AVG_ROW_LENGTH,
    DATA_LENGTH
    > FROM INFORMATION_SCHEMA.PARTITIONS
    > WHERE TABLE_SCHEMA = 'p' AND TABLE_NAME ='th';
+------------+----------------+------------+----------------+-------------+
| TABLE_NAME | PARTITION_NAME | TABLE_ROWS | AVG_ROW_LENGTH | DATA_LENGTH |
+------------+----------------+------------+----------------+-------------+
| th         | p0             |          2 |             20 |          20 |
| th         | p1             |          0 |              0 |           0 |
+------------+----------------+------------+----------------+-------------+
2 rows in set (0.00 sec)
```

Partition Management

- There are a number of ways using SQL statements to
  - modify partitioned tables;
  - it is possible to add, drop, redefine, merge, or split existing partitions using the partitioning extensions to the ALTER TABLE statement.
  - To change a table's partitioning scheme, it is necessary only to use the ALTER TABLE statement with a *partition_options* option.

```
CREATE TABLE trb3 (id INT, name VARCHAR(50), purchased DATE)
    PARTITION BY RANGE( YEAR(purchased) ) (
            PARTITION p0 VALUES LESS THAN (1990),
            PARTITION p1 VALUES LESS THAN (1995),
            PARTITION p2 VALUES LESS THAN (2000),
            PARTITION p3 VALUES LESS THAN (2005)
    );

ALTER TABLE trb3 PARTITION BY KEY(id) PARTITIONS 2;
```

Management of RANGE and LIST Partitions


```
mysql> CREATE TABLE tr (id INT, name VARCHAR(50), purchased DATE)
    -> PARTITION BY RANGE( YEAR(purchased) ) (
    -> PARTITION p0 VALUES LESS THAN (1990),
    -> PARTITION p1 VALUES LESS THAN (1995),
    -> PARTITION p2 VALUES LESS THAN (2000),
    -> PARTITION p3 VALUES LESS THAN (2005),
    -> PARTITION p4 VALUES LESS THAN (2010),
    -> PARTITION p5 VALUES LESS THAN (2015)
    -> );
Query OK, 0 rows affected (0.28 sec)
mysql> INSERT INTO tr VALUES
    -> (1, 'desk organiser', '2003-10-15'),
    -> (2, 'alarm clock', '1997-11-05'),
    -> (3, 'chair', '2009-03-10'),
    -> (4, 'bookcase', '1989-01-10'),
    -> (5, 'exercise bike', '2014-05-09'),
    -> (6, 'sofa', '1987-06-05'),
    -> (7, 'espresso maker', '2011-11-22'),
    -> (8, 'aquarium', '1992-08-04'),
    -> (9, 'study desk', '2006-09-16'),
    -> (10, 'lava lamp', '1998-12-25');
Query OK, 10 rows affected (0.05 sec) Records: 10 Duplicates: 0 Warnings: 0
```

Management of RANGE and LIST Partitions


```
mysql> SELECT * FROM tr
    -> WHERE purchased BETWEEN '1995-01-01' AND '1999-12-31';
+------+--------------+---------------+
|   id |         name |   purchased |
+------+--------------+---------------+
|    2 | alarm clock | 1997-11-05 |
|   10 |    lava lamp | 1998-12-25 |
+------+--------------+---------------+
2 rows in set (0.00 sec)

mysql> SELECT * FROM tr PARTITION (p2);
+------+--------------+---------------+
|   id |         name |   purchased |
+------+--------------+---------------+
|    2 | alarm clock | 1997-11-05 |
|   10 |    lava lamp | 1998-12-25 |
+------+--------------+---------------+
2 rows in set (0.00 sec)
```

Management of RANGE and LIST Partitions


```
mysql> ALTER TABLE tr DROP PARTITION p2;
Query OK, 0 rows affected (0.03 sec)
mysql> SELECT * FROM tr WHERE purchased
    -> BETWEEN '1995-01-01' AND '1999-12-31';
Empty set (0.00 sec)

mysql> SHOW CREATE TABLE tr\G
*************************** 1. row ***************************
       Table: tr
Create Table: CREATE TABLE `tr` (
    `id` int(11) DEFAULT NULL,
    `name` varchar(50) DEFAULT NULL,
    `purchased` date DEFAULT NULL
) ENGINE=InnoDB DEFAULT CHARSET=latin1
/*!50100 PARTITION BY RANGE ( YEAR(purchased))
(PARTITION p0 VALUES LESS THAN (1990) ENGINE = InnoDB,
 PARTITION p1 VALUES LESS THAN (1995) ENGINE = InnoDB,
 PARTITION p3 VALUES LESS THAN (2005) ENGINE = InnoDB,
 PARTITION p4 VALUES LESS THAN (2010) ENGINE = InnoDB,
 PARTITION p5 VALUES LESS THAN (2015) ENGINE = InnoDB) */
1 row in set (0.00 sec)
```

Management of RANGE and LIST Partitions


```
mysql> INSERT INTO tr VALUES (11, 'pencil holder', '1995-07-12');
Query OK, 1 row affected (0.00 sec)
mysql> SELECT * FROM tr WHERE purchased
    -> BETWEEN '1995-01-01' AND '2004-12-31';
+-----+------------------+---------------+
| id  |             name |  purchased    |
+-----+------------------+---------------+
|  1  | desk organizer   | 2003-10-15    |
| 11  |  pencil holder   | 1995-07-12    |
+-----+------------------+---------------+
2 rows in set (0.00 sec)
mysql> ALTER TABLE tr DROP PARTITION p3;
Query OK, 0 rows affected (0.03 sec)
mysql> SELECT * FROM tr WHERE purchased
    -> BETWEEN '1995-01-01' AND '2004-12-31';
Empty set (0.00 sec)
```

Management of RANGE and LIST Partitions


```
mysql> INSERT INTO tr VALUES (11, 'pencil holder', '1995-07-12');
Query OK, 1 row affected (0.00 sec)
mysql> SELECT * FROM tr WHERE purchased
    -> BETWEEN '1995-01-01' AND '2004-12-31';
+-----+------------------+---------------+
| id  |             name |    purchased  |
+-----+------------------+---------------+
|   1 | desk organizer | 2003-10-15 |
|  11 |   pencil holder | 1995-07-12 |
+-----+------------------+---------------+
2 rows in set (0.00 sec)
mysql> ALTER TABLE tr DROP PARTITION p3;
Query OK, 0 rows affected (0.03 sec)
mysql> SELECT * FROM tr WHERE purchased
    -> BETWEEN '1995-01-01' AND '2004-12-31';
Empty set (0.00 sec)
```

Management of RANGE and LIST Partitions


```
CREATE TABLE members (
        id INT,
        fname VARCHAR(25),
        lname VARCHAR(25),
        dob DATE
)
PARTITION BY RANGE( YEAR(dob) ) (
        PARTITION p0 VALUES LESS THAN (1980),
        PARTITION p1 VALUES LESS THAN (1990),
        PARTITION p2 VALUES LESS THAN (2000)
);

ALTER TABLE members ADD PARTITION (PARTITION p3 VALUES LESS THAN (2010));
```

Management of RANGE and LIST Partitions


```
mysql> ALTER TABLE members
    > ADD PARTITION (
    > PARTITION n VALUES LESS THAN (1970));
ERROR 1463 (HY000): VALUES LESS THAN value must be strictly » increasing for each partition

ALTER TABLE members
    REORGANIZE PARTITION p0 INTO (
        PARTITION n0 VALUES LESS THAN (1970),
        PARTITION n1 VALUES LESS THAN (1980)
);
```

Management of RANGE and LIST Partitions


```
CREATE TABLE tt (
        id INT,
        data INT
)
PARTITION BY LIST(data) (
        PARTITION p0 VALUES IN (5, 10, 15),
        PARTITION p1 VALUES IN (6, 12, 18)
);

ALTER TABLE tt ADD PARTITION (PARTITION p2 VALUES IN (7, 14, 21));

mysql> ALTER TABLE tt ADD PARTITION
        > (PARTITION np VALUES IN (4, 8, 12));
ERROR 1465 (HY000): Multiple definition of same constant » in list partitioning
```

Management of RANGE and LIST Partitions


```
CREATE TABLE employees (
        id INT NOT NULL,
        fname VARCHAR(50) NOT NULL,
        lname VARCHAR(50) NOT NULL,
        hired DATE NOT NULL
)
PARTITION BY RANGE( YEAR(hired) ) (
        PARTITION p1 VALUES LESS THAN (1991),
        PARTITION p2 VALUES LESS THAN (1996),
        PARTITION p3 VALUES LESS THAN (2001),
        PARTITION p4 VALUES LESS THAN (2005)
);
ALTER TABLE employees ADD PARTITION (
        PARTITION p5 VALUES LESS THAN (2010),
        PARTITION p6 VALUES LESS THAN MAXVALUE
);
```

Management of RANGE and LIST Partitions


```
ALTER TABLE members REORGANIZE PARTITION n0 INTO (
        PARTITION s0 VALUES LESS THAN (1960),
        PARTITION s1 VALUES LESS THAN (1970)
);

ALTER TABLE members REORGANIZE PARTITION s0,s1 INTO (
        PARTITION p0 VALUES LESS THAN (1970)
);

ALTER TABLE members REORGANIZE PARTITION p0,p1,p2,p3 INTO (
        PARTITION m0 VALUES LESS THAN (1980),
        PARTITION m1 VALUES LESS THAN (2000)
);

ALTER TABLE tt ADD PARTITION (PARTITION np VALUES IN (4, 8));
ALTER TABLE tt REORGANIZE PARTITION p1,np INTO (
        PARTITION p1 VALUES IN (6, 18),
        PARTITION np VALUES in (4, 8, 12)
);
```

Management of HASH and KEY Partitions

- You cannot drop partitions from tables that are partitioned by HASH or KEY in the same way that you can from tables that are partitioned by RANGE or LIST.
  - However, you can merge HASH or KEY partitions using ALTER TABLE ... COALESCE PARTITION.

```
CREATE TABLE clients (
    id INT,
    fname VARCHAR(30),
    lname VARCHAR(30),
    signed DATE
)
PARTITION BY HASH( MONTH(signed) )
PARTITIONS 12;

mysql> ALTER TABLE clients COALESCE PARTITION 4;
Query OK, 0 rows affected (0.02 sec)

ALTER TABLE clients ADD PARTITION PARTITIONS 6;
```

Exchanging Partitions and Subpartitions with Tables

- In MySQL 8.0, it is possible to exchange a table partition or subpartition with a table using

  ALTER TABLE *pt* EXCHANGE PARTITION *p* WITH TABLE *nt*,

  – where *pt* is the partitioned table and *p* is the partition or subpartition of *pt* to be exchanged with unpartitioned table *nt*, provided that the following statements are true:
    - Table *nt* is not itself partitioned.
    - Table *nt* is not a temporary table.
    - The structures of tables *pt* and *nt* are otherwise identical.
    - Table nt contains no foreign key references, and no other table has any foreign keys that refer to nt.
    - There are no rows in *nt* that lie outside the boundaries of the partition definition for *p*. This condition does not apply if WITHOUT VALIDATION is used.
    - For InnoDB tables, both tables use the same row format. To determine the row format of an InnoDB table, query INFORMATION_SCHEMA.INNODB_TABLES.
    - nt does not have any partitions that use the DATA DIRECTORY option. This restriction is lifted for InnoDB tables in MySQL 8.0.14 and later.

Exchanging Partitions and Subpartitions with Tables


```
CREATE TABLE e (
        id INT NOT NULL,
        fname VARCHAR(30),
        lname VARCHAR(30)
)
PARTITION BY RANGE (id) (
        PARTITION p0 VALUES LESS THAN (50),
        PARTITION p1 VALUES LESS THAN (100),
        PARTITION p2 VALUES LESS THAN (150),
        PARTITION p3 VALUES LESS THAN (MAXVALUE)
);

INSERT INTO e VALUES
        (1669, "Jim", "Smith"),
        (337, "Mary", "Jones"),
        (16, "Frank", "White"),
        (2005, "Linda", "Black");
```

Exchanging Partitions and Subpartitions with Tables


```
mysql> CREATE TABLE e2 LIKE e;
Query OK, 0 rows affected (0.04 sec)
mysql> ALTER TABLE e2 REMOVE PARTITIONING;
Query OK, 0 rows affected (0.07 sec)
Records: 0 Duplicates: 0 Warnings: 0

mysql> SELECT PARTITION_NAME, TABLE_ROWS
                  FROM INFORMATION_SCHEMA.PARTITIONS
                  WHERE TABLE_NAME = 'e';
+----------------+------------+
| PARTITION_NAME | TABLE_ROWS |
+----------------+------------+
|             p0 |          1 |
|             p1 |          0 |
|             p2 |          0 |
|             p3 |          3 |
+----------------+------------+
2 rows in set (0.00 sec)
```

Exchanging Partitions and Subpartitions with Tables


```
mysql> ALTER TABLE e EXCHANGE PARTITION p0 WITH TABLE e2;
Query OK, 0 rows affected (0.04 sec)

mysql> SELECT PARTITION_NAME, TABLE_ROWS
                    FROM INFORMATION_SCHEMA.PARTITIONS
                    WHERE TABLE_NAME = 'e';
+----------------+------------+
| PARTITION_NAME | TABLE_ROWS |
+----------------+------------+
|             p0 |          0 |
|             p1 |          0 |
|             p2 |          0 |
|             p3 |          3 |
+----------------+------------+
4 rows in set (0.00 sec)

mysql> SELECT * FROM e2;
+-----+---------+---------+
| id  | fname   | lname   |
+-----+---------+---------+
| 16  |  Frank  | White   |
+-----+---------+---------+
1 row in set (0.00 sec)
```

Exchanging Partitions and Subpartitions with Tables

- **Nonmatching Rows**

```
mysql> INSERT INTO e2 VALUES (51, "Ellen", "McDonald");
Query OK, 1 row affected (0.08 sec)
mysql> ALTER TABLE e EXCHANGE PARTITION p0 WITH TABLE e2;
ERROR 1707 (HY000): Found row that does not match the partition

mysql> ALTER TABLE e EXCHANGE PARTITION p0 WITH TABLE e2 WITHOUT VALIDATION;
Query OK, 0 rows affected (0.02 sec)
```

Maintenance of Partitions

- Rebuilding partitions
  - ALTER TABLE t1 REBUILD PARTITION p0, p1;
- Optimizing partitions
  - ALTER TABLE t1 OPTIMIZE PARTITION p0, p1;
- Analyzing partitions
  - ALTER TABLE t1 ANALYZE PARTITION p3;
- Repairing partitions
  - ALTER TABLE t1 REPAIR PARTITION p0,p1;
- Checking partitions
  - ALTER TABLE trb3 CHECK PARTITION p1;

Partition Pruning

- "cutting away" of unneeded partitions is known as pruning

```
CREATE TABLE t1 (
        fname VARCHAR(50) NOT NULL,
        lname VARCHAR(50) NOT NULL,
        region_code TINYINT UNSIGNED NOT NULL,
        dob DATE NOT NULL
)
PARTITION BY RANGE( region_code ) (
        PARTITION p0 VALUES LESS THAN (64),
        PARTITION p1 VALUES LESS THAN (128),
        PARTITION p2 VALUES LESS THAN (192),
        PARTITION p3 VALUES LESS THAN MAXVALUE
);

SELECT fname, lname, region_code, dob
        FROM t1
        WHERE region_code > 125 AND region_code < 130;
```

Partition Pruning

- "cutting away" of unneeded partitions is known as pruning

```
CREATE TABLE t1 (
        fname VARCHAR(50) NOT NULL,
        lname VARCHAR(50) NOT NULL,
        region_code TINYINT UNSIGNED NOT NULL,
        dob DATE NOT NULL
)
PARTITION BY RANGE( region_code ) (
        PARTITION p0 VALUES LESS THAN (64),
        PARTITION p1 VALUES LESS THAN (128),
        PARTITION p2 VALUES LESS THAN (192),
        PARTITION p3 VALUES LESS THAN MAXVALUE
);

SELECT fname, lname, region_code, dob
        FROM t1
        WHERE region_code > 125 AND region_code < 130;
```

Partition Selection

- Partition selection is similar to partition pruning,
  - in that only specific partitions are checked for matches, but differs in two key respects:
    - The partitions to be checked are specified by the issuer of the statement, unlike partition pruning, which is automatic.
    - Whereas partition pruning applies only to queries, explicit selection of partitions is supported for both queries and a number of DML statements.
- SQL statements supporting explicit partition selection are listed here:
  - SELECT
  - DELETE
  - INSERT
  - REPLACE
  - UPDATE
  - LOAD DATA.
  - LOAD XML.

作业七

- 请你根据上课内容，针对你在E-BookStore项目中的数据库设计，详细回答下列问题：
  1. 请你详细描述如何通过全量备份和增量备份来实现系统状态恢复。(2分)
  2. 请你根据MySQL缓存的工作原理，描述预取机制的优点。(1分)
  3. 请你按照你的理解，阐述Partition机制有什么好处？如果数据文件在一台机器上有足够的存储空间存储，是否还需要进行Partition？(2分)

  – 请提交包含上述问题答案的文档，答案应该结合你的E-BookStore的具体设计来阐述，不要过于泛化。

- 评分标准：
  – 分值如问题描述，答案不唯一，只要你的说理合理即可视为正确。

Thank You!