分区表

one2c———采集versa数据流量的时候可以这么优化

官网：https://dev.mysql.com/doc/refman/5.7/en/partitioning-types.html

——目的，减少IO量

mysql会按照自己定义的规则，会把一个数据库文件存储成 n 多个 不同类型的文件，可以按照某一个列进行分区，也可以按照数据的一个范围进行分区，也可以按照hash散列进行分区等。相同类型的数据会聚集在一个文件中，这样在查询的时候会判断是在哪一个文件里面，然后只在当前文件中进行查找，而不需要扫描所有的数据——分而治之的思想 。

对于用户而言，分区表是一个独立的逻辑表，但是底层是由多个物理子表组成。分区表对于用户而言是一个完全封装底层实现的黑盒子，对用户而言是透明的，从文件系统中可以看到多个使用#分隔命名的表文件。

mysql在创建表时使用partition by子句定义每个分区存放的数据，在执行查询的时候，优化器会根据分区定义过滤那些没有我们需要数据的分区，这样查询就无须扫描所有分区。

分区的主要目的是将数据安好一个较粗的力度分在不同的表中，这样可以将相关的数据存放在一起。

▼ 分区表的应用场景

- 表非常大以至于无法全部都放在内存中，或者只在表的最后部分有热点数据，其他均是历史数据
- 分区表的数据更容易维护

  批量删除大量数据可以使用清除整个分区的方式

  对一个独立分区进行优化、检查、修复等操作
- 分区表的数据可以分布在不同的物理设备上，从而高效地利用多个硬件设备
- 可以使用分区表来避免某些特殊的瓶颈

  innodb的单个索引的互斥访问

  ——innodb对数据加锁是给索引加锁，分区表可以减少互斥操作

  ext3文件系统的inode锁竞争
- 可以备份和恢复独立的分区

▼ 分区表的限制

- 一个表最多只能有1024个分区，在5.7版本的时候可以支持8196个分区
- 在早期的mysql中，分区表达式必须是整数或者是返回整数的表达式，在mysql5.5中，某些场景可以直接使用列来进行分区
- 如果分区字段中有主键或者唯一索引的列，那么所有主键列和唯一索引列都必须包含进来
- 分区表无法使用外键约束

▼ 分区表的原理

分区表由多个相关的底层表实现，这个底层表也是由句柄对象标识，我们可以直接访问各个分区。存储引擎管理分区的各个底层表和管理普通表一样（所有的底层表都必须使用相同的存储引擎），分区表的索引知识在各个底层表上各自加上一个完全相同的索引。从存储引擎的角度来看，底层表和普通表没有任何不同，存储引擎也无须知道这是一个普通表还是一个分区表的一部分。

分区表的操作按照以下的操作逻辑进行：

select查询

当查询一个分区表的时候，分区层先打开并锁住所有的底层表，优化器先判断是否可以过滤部分分区，然后再调用对应的存储引擎接口访问各个分区的数据

insert操作

当写入一条记录的时候，分区层先打开并锁住所有的底层表，然后确定哪个分区接受这条记录，再将记录写入对应底层表

delete操作

当删除一条记录时，分区层先打开并锁住所有的底层表，然后确定数据对应的分区，最后对相应底层表进行删除操作

update操作

当更新一条记录时，分区层先打开并锁住所有的底层表，mysql先确定需要更新的记录再哪个分区，然后取出数据并更新，再判断更新后的数据应该再哪个分区，最后对底层表进行写入操作，并对源数据所在的底层表进行删除操作

有些操作时支持过滤的，例如，当删除一条记录时，MySQL需要先找到这条记录，如果where条件恰好和分区表达式匹配，就可以将所有不包含这条记录的分区都过滤掉，这对update同样有效。如果是insert操作，则本身就是只命中一个分区，其他分区都会被过滤掉。mysql先确定这条记录属于哪个分区，再将记录写入对应得曾分区表，无须对任何其他分区进行操作

虽然每个操作都会“先打开并锁住所有的底层表”，但这并不是说分区表在处理过程中是锁住全表的，如果存储引擎能够自己实现行级锁，例如innodb，则会在分区层释放对应表锁。

▼ 分区表的类型

▼ 范围分区

根据列值在给定范围内将行分配给分区

范围分区表的分区方式是：每个分区都包含行数据且分区的表达式在给定的范围内，分区的范围应该是连续的且不能重叠，可以使用values less than运算符来定义。

1、创建普通的表

```
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
);
```

2、创建带分区的表，下面建表的语句是按照store_id来进行分区的，指定了4个分区

```
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)
);
--在当前的建表语句中可以看到，store_id的值在1-5的在p0分区，6-10的在p1分区，11-15的在p3分区，16-20的在p4分区，但是如果插入超过20的值就会报错，因为mysq
```

3、可以使用less than maxvalue来避免此种情况

```
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
);
--maxvalue表示始终大于等于最大可能整数值的整数值
```

4、可以使用相同的方式根据员工的职务代码对表进行分区

```
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)
);
```

5、可以使用date类型进行分区：如虚妄根据每个员工离开公司的年份进行划分，如year(separated)

```
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 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
);
```

6、可以使用函数根据range的值来对表进行分区，如timestampunix_timestamp()

```
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)
);
--timestamp不允许使用任何其他涉及值的表达式
```

基于时间间隔的分区方案，在mysql5.7中，可以基于范围或事件间隔实现分区方案，有两种选择

1、基于范围的分区，对于分区表达式，可以使用操作函数基于date、time、或者datatime列来返回一个整数值

```
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
);

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)
);
```

2、基于范围列的分区，使用date或者datatime列作为分区列

```
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 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
);
```

▼ 列表分区

类似于按range分区，区别在于list分区是基于列值匹配一个离散值集合中的某个值来进行选择

```
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)
);
```

▼ 列分区

mysql从5.5开始支持column分区，可以认为i是range和list的升级版，在5.5之后，可以使用column分区替代range和list，但是column分区只接受普通列不接受表达式

```
CREATE TABLE `list_c` (
 `c1` int(11) DEFAULT NULL,
 `c2` int(11) DEFAULT NULL
) ENGINE=InnoDB DEFAULT CHARSET=latin1
/*!50500 PARTITION BY RANGE COLUMNS(c1)
(PARTITION p0 VALUES LESS THAN (5) ENGINE = InnoDB,
```

```
 PARTITION p1 VALUES LESS THAN (10) ENGINE = InnoDB) */
 CREATE TABLE `list_c` (
 `c1` int(11) DEFAULT NULL,
 `c2` int(11) DEFAULT NULL,
 `c3` char(20) DEFAULT NULL
) ENGINE=InnoDB DEFAULT CHARSET=latin1
/*!50500 PARTITION BY RANGE COLUMNS(c1,c3)
(PARTITION p0 VALUES LESS THAN (5,'aaa') ENGINE = InnoDB,
 PARTITION p1 VALUES LESS THAN (10,'bbb') ENGINE = InnoDB) */
 CREATE TABLE `list_c` (
 `c1` int(11) DEFAULT NULL,
 `c2` int(11) DEFAULT NULL,
 `c3` char(20) DEFAULT NULL
) ENGINE=InnoDB DEFAULT CHARSET=latin1
/*!50500 PARTITION BY LIST COLUMNS(c3)
(PARTITION p0 VALUES IN ('aaa') ENGINE = InnoDB,
 PARTITION p1 VALUES IN ('bbb') ENGINE = InnoDB) */
```

▼ hash分区

基于用户定义的表达式的返回值来进行选择的分区，该表达式使用将要插入到表中的这些行的列值进行计算。这个函数可以包含myql中有效的、产生非负整数值的任何表达式

```
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;    ---根据store_id  hash 运算  最后分四个分区


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;
```

▼ key 分区

类似于hash分区，区别在于key分区只支持一列或多列，且mysql服务器提供其自身的哈希函数，必须有一列或多列包含整数值

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

▼ 子分区

在分区的基础之上，再进行分区后存储

```
CREATE TABLE `t_partition_by_subpart`
(
  `id` INT AUTO_INCREMENT,
  `sName` VARCHAR(10) NOT NULL,
  `sAge` INT(2) UNSIGNED ZEROFILL NOT NULL,
  `sAddr` VARCHAR(20) DEFAULT NULL,
```

```
  `sGrade` INT(2) NOT NULL,
  `sStuId` INT(8) DEFAULT NULL,
  `sSex` INT(1) UNSIGNED DEFAULT NULL,
  PRIMARY KEY (`id`, `sGrade`)
)  ENGINE = INNODB
PARTITION BY RANGE(id)
SUBPARTITION BY HASH(sGrade) SUBPARTITIONS 2
(
PARTITION p0 VALUES LESS THAN(5),
PARTITION p1 VALUES LESS THAN(10),
PARTITION p2 VALUES LESS THAN(15)
);
```

▼ 如何使用分区表

如果需要从非常大的表中查询出某一段时间的记录，而这张表中包含很多年的历史数据，数据是按照时间排序的，此时应该如何查询数据呢？

因为数据量巨大，肯定不能在每次查询的时候都扫描全表。考虑到索引在空间和维护上的消耗，也不希望使用索引，即使使用索引，会发现会产生大量的碎片，还会产生大量的随机IO，但是当数据量超大的时候，索引也就无法起作用了，此时可以考虑使用分区来进行解决

- 全量扫描数据，不要任何索引

  使用简单的分区方式存放表，不要任何索引，根据分区规则大致定位需要的数据为止，通过使用where条件将需要的数据限制在少数分区中，这种策略适用于以正常的方式访问大量数据

- 索引数据，并分离热点

  如果数据有明显的热点（经常查询的数据），而且除了这部分数据，其他数据很少被访问到，那么可以将这部分热点数据单独放在一个分区中，让这个分区的数据能够有机会都缓存在内存中，这样查询就可以只访问一个很小的分区表，能够使用索引，也能够有效的使用缓存

▼ 在使用分区表的时候需要注意的问题

- null值会使分区过滤无效
- 分区列和索引列不匹配，会导致查询无法进行分区过滤
- 选择分区的成本可能很高
- 打开并锁住所有底层表的成本可能很高
- 维护分区的成本可能很高