INDEXING

Lê Hồng Hải
UET-VNUH

Today's Overview

1 Index

2 Compound index

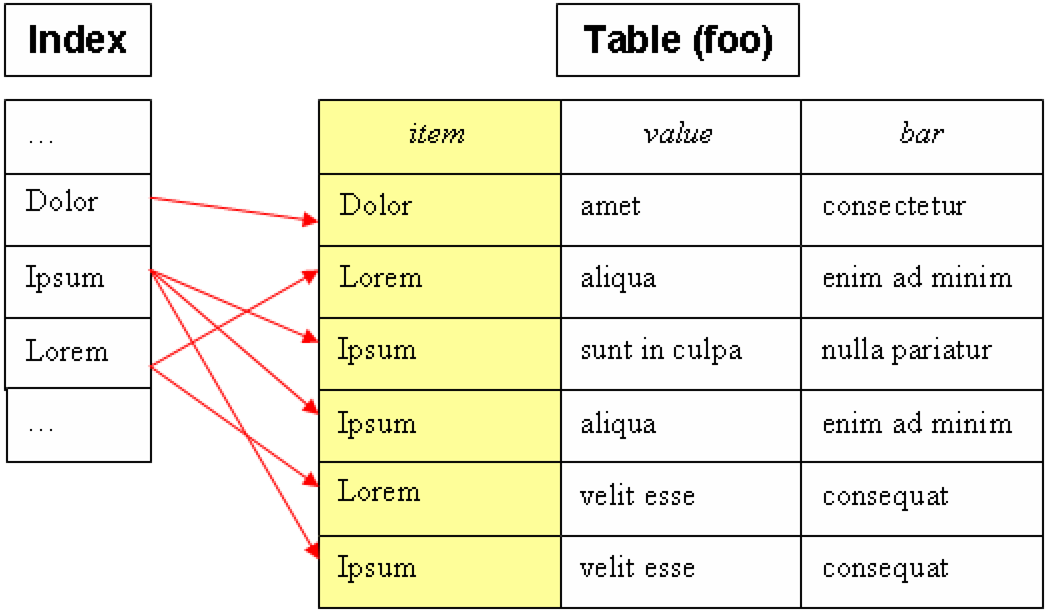
3 Function Index

4 FTS

Indexing

- Using indexes to quickly find rows with specific column values
- Without an index, MySQL must scan the whole table to locate the relevant rows

The larger table, the slower it searches



Index

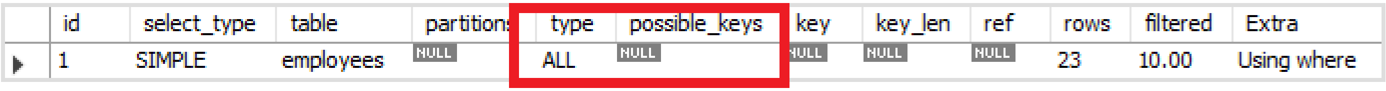
...
Dolor
Ipsum
Lorem
...

Table (foo)

item	*value*	*bar*
Dolor	amet	consectetur
Lorem	aliqua	enim ad minim
Ipsum	sunt in culpa	nulla pariatur
Ipsum	aliqua	enim ad minim
Lorem	velit esse	consequat
Ipsum	velit esse	consequat

Query Explain example

```
Explain SELECT
    employeeNumber,
    lastName,
    firstName
FROM
    employees
WHERE
    jobTitle = 'Sales Rep’;
```



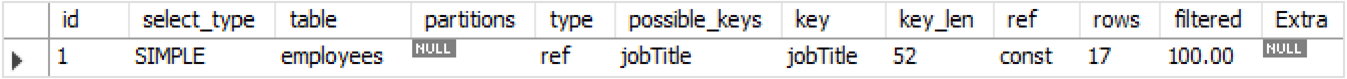
	id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
▶	1	SIMPLE	employees	NULL	ALL	NULL	NULL	NULL	NULL	23	10.00	Using where

- MySQL had to scan the whole table to find the employees with the Sales Rep job title

Create Index

- Let’s create an index for the jobTitle column by using the CREATE INDEX statement:
  - **CREATE INDEX jobTitle ON employees(jobTitle);**

```
EXPLAIN SELECT
    employeeNumber,
    lastName,
    firstName
FROM
    employees
WHERE
    jobTitle = 'Sales Rep';
```



	id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
▶	1	SIMPLE	employees	NULL	ref	jobTitle	jobTitle	52	const	17	100.00	NULL

Indexing LIKE Filters

- There are search terms that can be indexed very well, but others can not
- It is the position of the wild card characters that make all the difference
- Can not be indexed
  - SELECT * FROM tbl_name WHERE key_col LIKE '%Patrick%’;
- Can be indexed
  - SELECT * FROM tbl_name WHERE key_col LIKE 'Patrick%’
  - SELECT * FROM tbl_name WHERE key_col LIKE 'Pat%_ck%';

Compound Index

A compound (composite) index is an index on multiple columns.

```
CREATE TABLE Employee (
   id INT NOT NULL,
   lastname varchar(50) not null,
   firstname varchar(50) not null,
   PRIMARY KEY (id),
   INDEX name (lastname, firstname)
);
```

Compound Index

- The query optimizer uses the composite indexes for queries that:
  - Test all columns in the index, or
  - Test the first columns, the first two columns, and so on

Compound Index

- The following queries use the name index:

  SELECT * FROM Employee WHERE lastname='Shah';

  SELECT * FROM Employee WHERE lastname ='Shah' AND firstname ='Mona’

- There are some queries in which composite indexes will not work:

  SELECT * FROM Employee WHERE firstname='Mona';

  SELECT * FROM Employee WHERE lastname='Shah' OR firstname='Mona';

Primary Index

- When you create a table with a primary key or unique key, MySQL automatically creates a special index named PRIMARY. This index is called the clustered index.

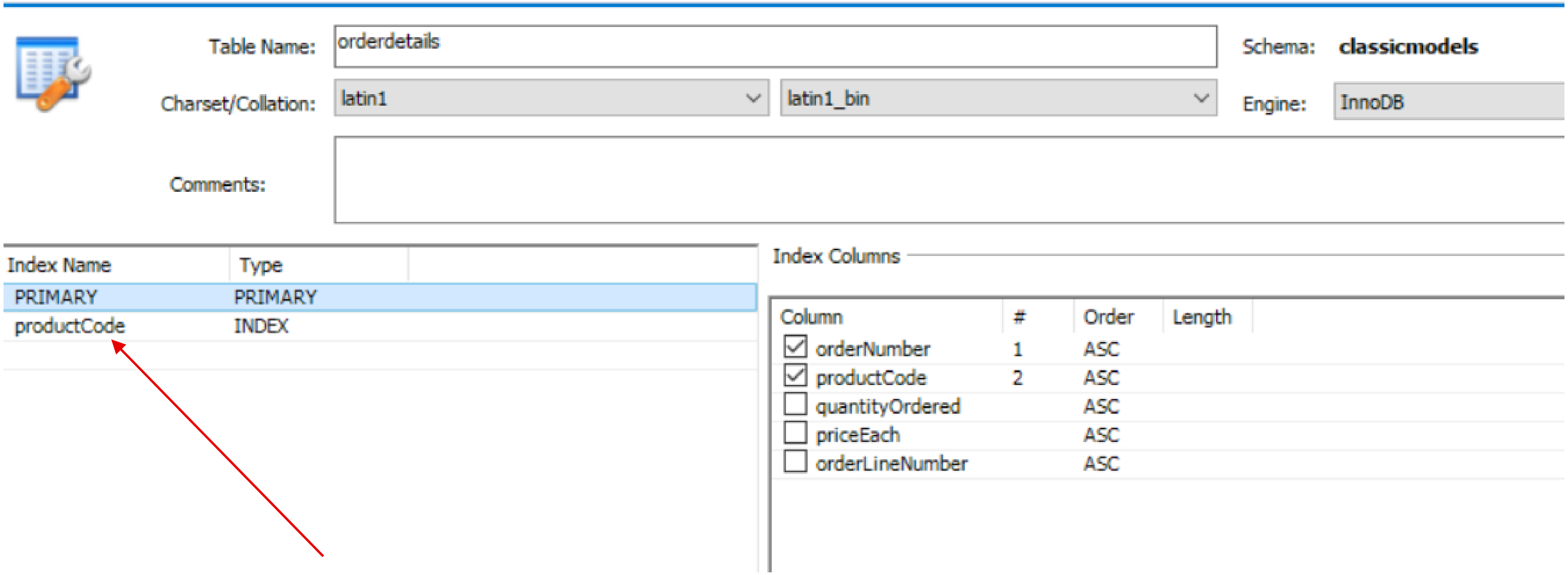


Table Name: orderdetails

Schema: classicmodels

Charset/Collation: latin1 | latin1_bin

Engine: InnoDB

Comments:

Index Name	Type
PRIMARY	PRIMARY
productCode	INDEX

Index Columns

Column	#	Order	Length
☑ orderNumber	1	ASC	
☑ productCode	2	ASC	
□ quantityOrdered		ASC	
□ priceEach		ASC	
□ orderLineNumber		ASC	

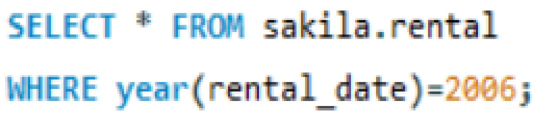
Why do we need to create ‘productCode’ index?

Descending Index

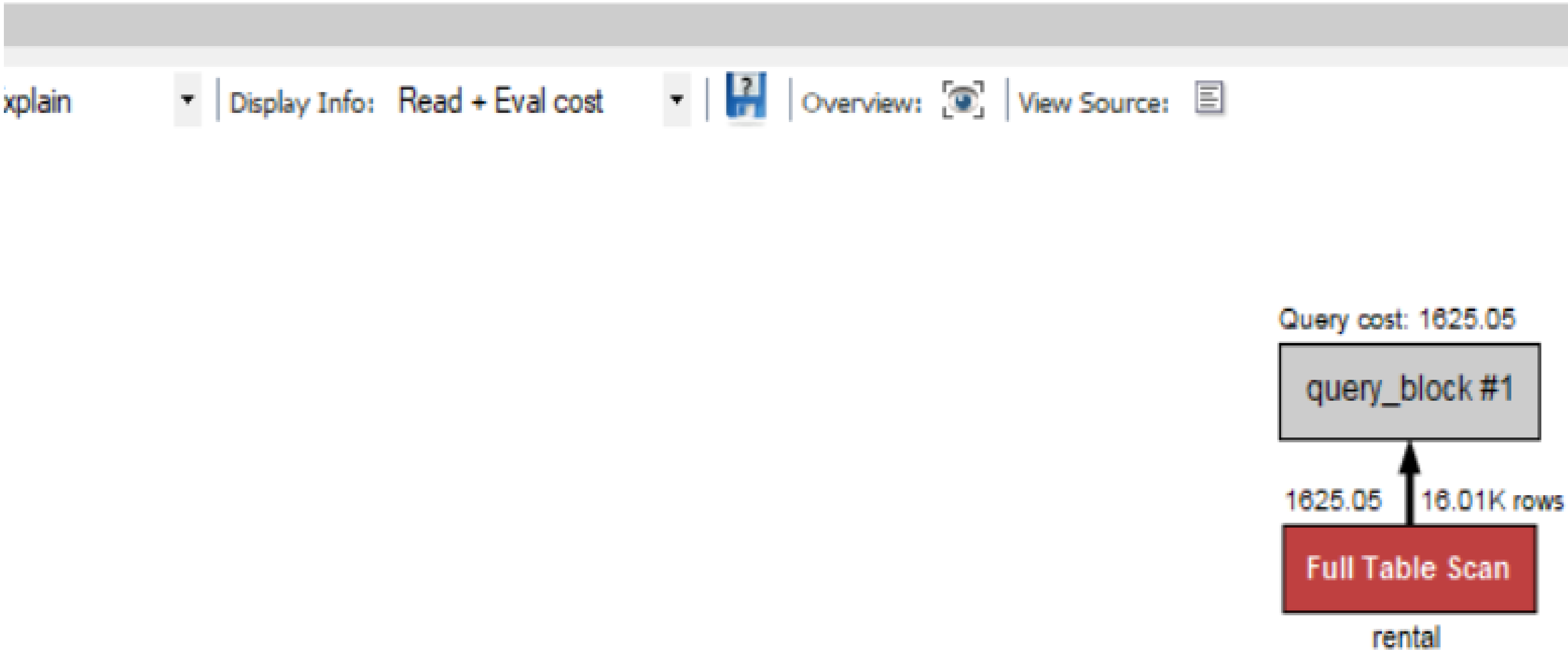
- A descending index is an index that stores key values in descending order

```
EXPLAIN SELECT
    *
FROM
    t
ORDER BY a DESC , b DESC; -- use index a_desc_b_desc
```

Function Index



```
SELECT * FROM sakila.rental
WHERE year(rental_date)=2006;
```



xplain

Display Info: Read + Eval cost

Function Index

- From MySQL 8.0.13, there is support for indexing using functions

```
 Alter table orders add index
((year(orderDate)), (month(orderDate)));
```

FULLTEXT Search (FTS)

- Full-text search is a technique to search for documents that don’t perfectly match the search criteria
- For example, you can search for Wood and Metal, FTS can return results that contain the searched words separately

LIKE Filter

- MySQL has to scan the whole table to find the exact text based on a pattern in the LIKE statement or pattern in the regular expressions
- *It is difficult to have a flexible search query*

Create FULLTEXT index

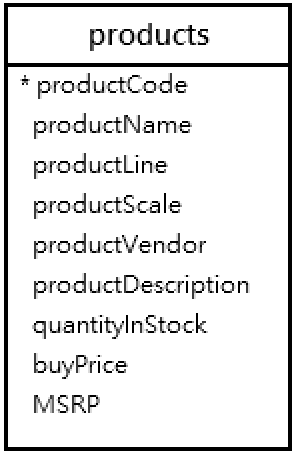
```
CREATE TABLE table_name(
    column_list,
    ...,
    FULLTEXT (column1,column2,..)
);
```

Create FULLTEXT Search

- Create a full-text search in the productLine column of the products table using the ALTER TABLE ADD FULLTEXT statement:

```
ALTER TABLE products

ADD FULLTEXT(productline);
```



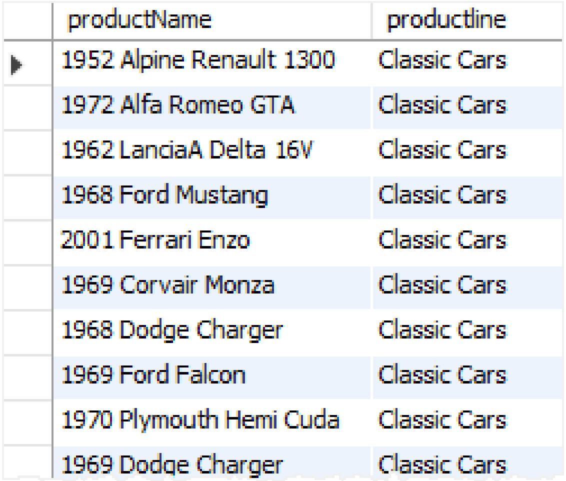
products

* productCode
productName
productLine
productScale
productVendor
productDescription
quantityInStock
buyPrice
MSRP

Search using FTS

- You can search for products whose product lines contain the term Classic . You use the MATCH() and AGAINST() functions as the following query:

```
SELECT
    productName,
    productLine
FROM products
WHERE
    MATCH(productLine)
    AGAINST('Classic');
```

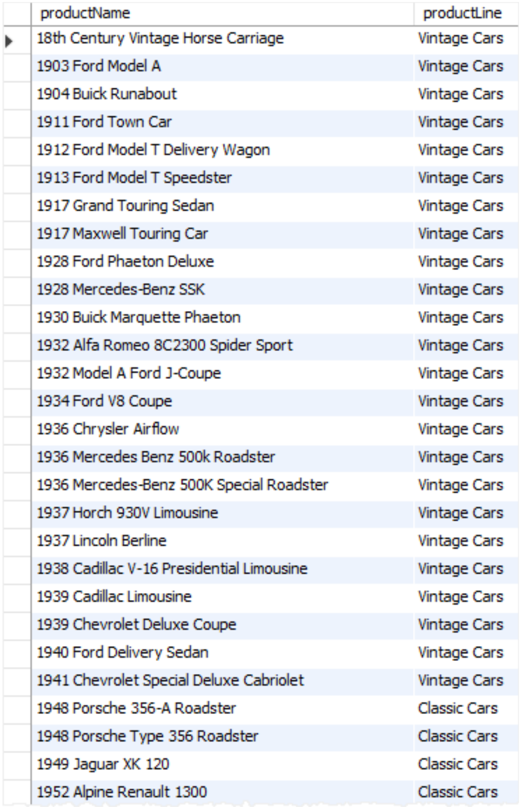


productName	productline
1952 Alpine Renault 1300	Classic Cars
1972 Alfa Romeo GTA	Classic Cars
1962 LanciaA Delta 16V	Classic Cars
1968 Ford Mustang	Classic Cars
2001 Ferrari Enzo	Classic Cars
1969 Corvair Monza	Classic Cars
1968 Dodge Charger	Classic Cars
1969 Ford Falcon	Classic Cars
1970 Plymouth Hemi Cuda	Classic Cars
1969 Dodge Charger	Classic Cars

Search using FTS

- To search for a product whose product line contains Classic or Vintage term, you can use the following query:

```
SELECT
    productName,
     productLine
FROM products
WHERE
    MATCH(productline)
     AGAINST('Classic,Vintage')
ORDER BY productName;
```



productName	productLine
18th Century Vintage Horse Carriage	Vintage Cars
1903 Ford Model A	Vintage Cars
1904 Buick Runabout	Vintage Cars
1911 Ford Town Car	Vintage Cars
1912 Ford Model T Delivery Wagon	Vintage Cars
1913 Ford Model T Speedster	Vintage Cars
1917 Grand Touring Sedan	Vintage Cars
1917 Maxwell Touring Car	Vintage Cars
1928 Ford Phaeton Deluxe	Vintage Cars
1928 Mercedes-Benz SSK	Vintage Cars
1930 Buick Marquette Phaeton	Vintage Cars
1932 Alfa Romeo 8C2300 Spider Sport	Vintage Cars
1932 Model A Ford J-Coupe	Vintage Cars
1934 Ford V8 Coupe	Vintage Cars
1936 Chrysler Airflow	Vintage Cars
1936 Mercedes Benz 500k Roadster	Vintage Cars
1936 Mercedes-Benz 500K Special Roadster	Vintage Cars
1937 Horch 930V Limousine	Vintage Cars
1937 Lincoln Berline	Vintage Cars
1938 Cadillac V-16 Presidential Limousine	Vintage Cars
1939 Cadillac Limousine	Vintage Cars
1939 Chevrolet Deluxe Coupe	Vintage Cars
1940 Ford Delivery Sedan	Vintage Cars
1941 Chevrolet Special Deluxe Cabriolet	Vintage Cars
1948 Porsche 356-A Roadster	Classic Cars
1948 Porsche Type 356 Roadster	Classic Cars
1949 Jaguar XK 120	Classic Cars
1952 Alpine Renault 1300	Classic Cars

Boolean Full-Text Searches

- In the Boolean mode, MySQL searches for words instead of the *concept* like in the natural language search

```
SELECT productName, productline
FROM products
WHERE MATCH(productName) AGAINST('Truck -
Pickup' IN BOOLEAN MODE )
```

Boolean Full-Text Searches

- To search for rows that contain at least one of the two words: mysql or tutorial

  ‘mysql tutorial’

- To search for rows that contain both words: mysql and tutorial

  ‘+mysql +tutorial’

- To search for rows that contain the word “mysql”, but put the higher rank for the rows that contain “tutorial”:

  ‘+mysql tutorial’

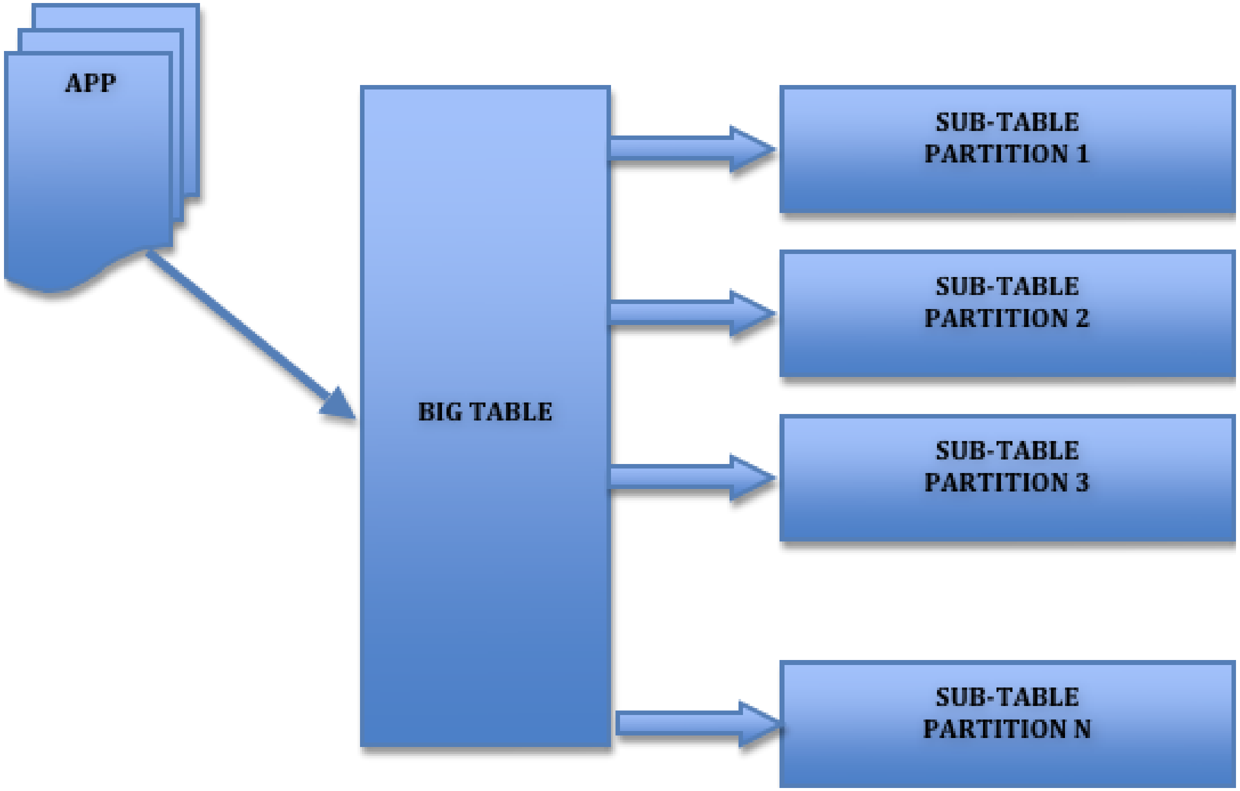
…

ngram Full-Text Parser

- When it comes to ideographic languages such as Chinese, Japanese, and Korean, the full-text parser has a limitation in that these ideographic languages do not use word delimiters
- MySQL provided the ngram full-text parser. Since version 5.7.6, MySQL included ngram full-text parser as a built-in server plugin delimiters

PARTITIONING

PARTITIONING



PARTITIONING

- Parts of the table are saved as separate tables in different locations
- Allows distribution of table parts across the file system according to established rules (partitioning function)

Advantages

- Some queries may be optimal if the data that satisfies the WHERE clause is determined to be stored in one or more partitions
- You can also use partitioning to distribute the data across different disks
- Partitions are updateable, data can be reorganized to enhance frequent queries
- Data that is no longer useful can often be easily removed by deleting the partition

Partitioning types

- RANGE: assigns rows to partitions based on column values within a range
- LIST: similar to RANGE, but the list is a collection of discrete values
- HASH: based on the value returned by a user-defined expression (produces an integer, non-negative value)
- KEY: similar to hash partitioning, except that the hash function is provided by the MySQL server

Key Partitioning

```
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;
```

Range Partitioning

```
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
     );
```

List Partitioning

```
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)
);
```

Partition Management

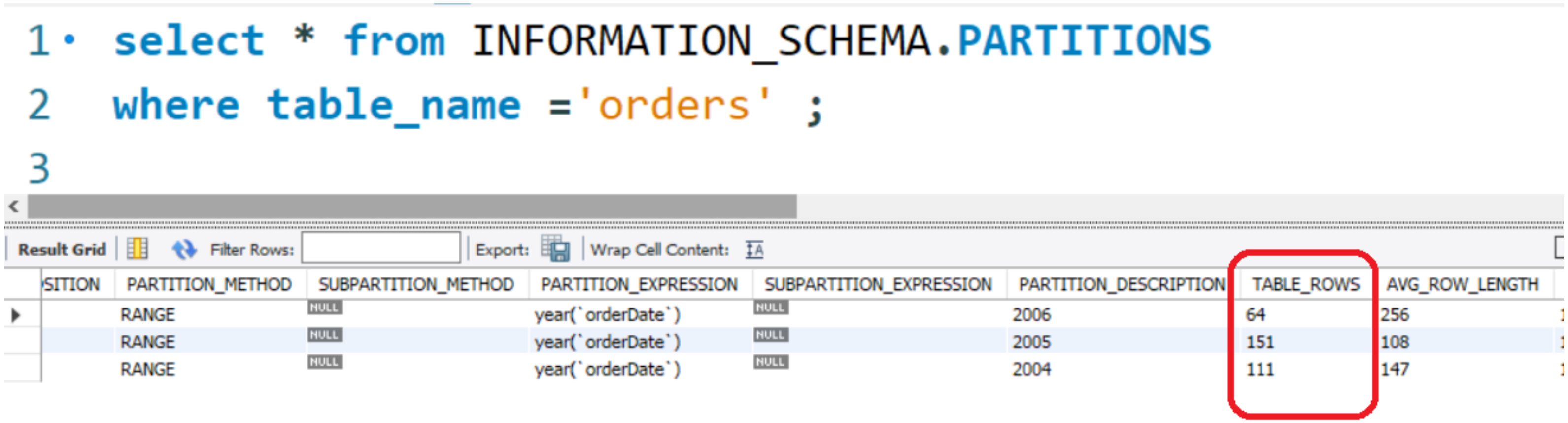
- ALTER TABLE
  - PARTITION BY, ADD PARTITION, DROP PARTITION, REORGANIZE PARTITION, COALESCE PARTITION

Partition Management

- ALTER TABLE *trb3* PARTITION BY KEY(id) PARTITIONS 2;
- ALTER TABLE *tr* DROP PARTITION p2;
- ALTER TABLE ADD PARTITION (PARTITION p3 VALUES LESS THAN (2000));

Partition Information

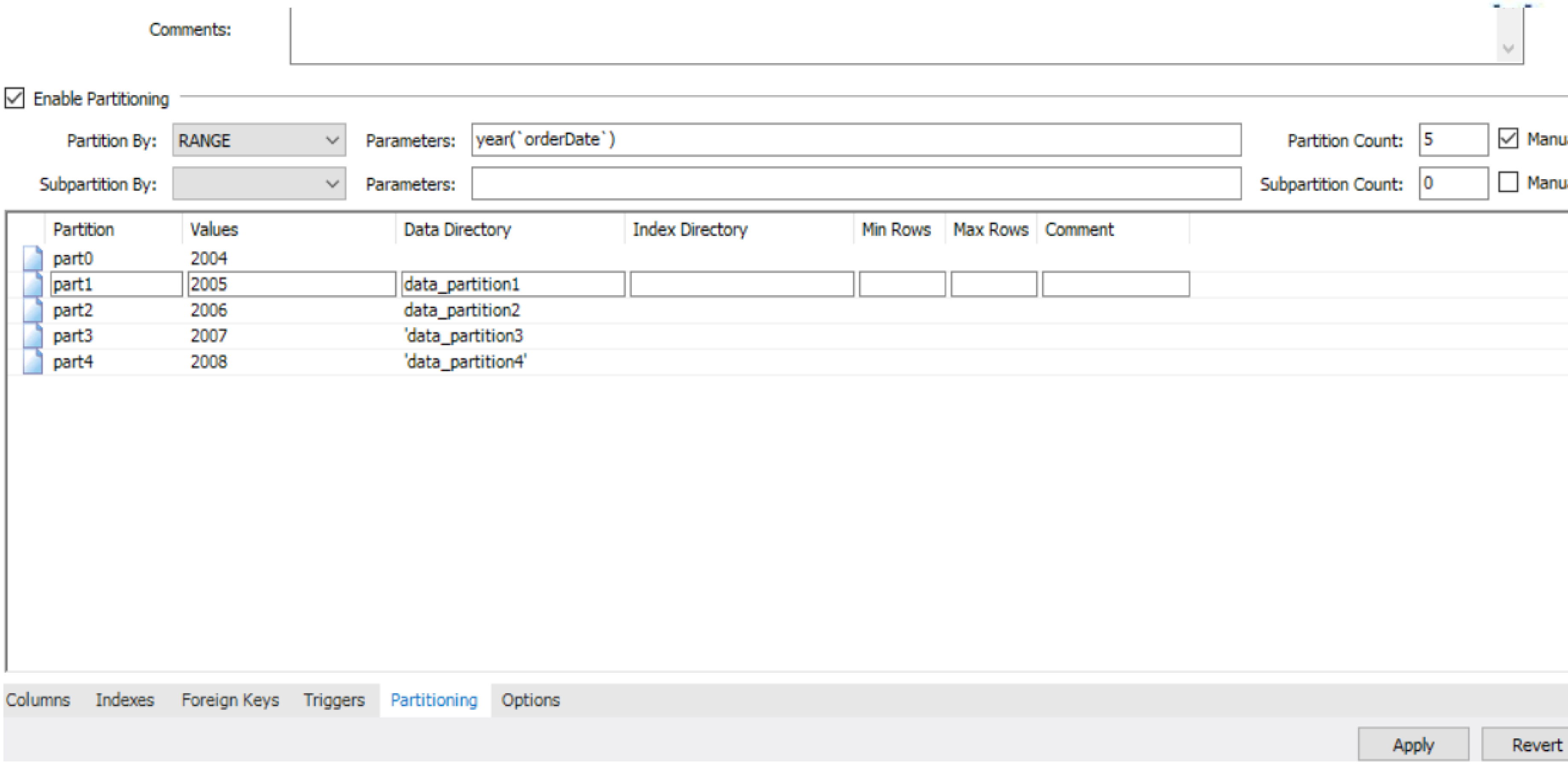
- SHOW CREATE TABLE
- SHOW TABLE STATUS
- **INFORMATION_SCHEMA.PARTITIONS**



```
select * from INFORMATION_SCHEMA.PARTITIONS
where table_name ='orders' ;
```

SITION	PARTITION_METHOD	SUBPARTITION_METHOD	PARTITION_EXPRESSION	SUBPARTITION_EXPRESSION	PARTITION_DESCRIPTION	TABLE_ROWS	AVG_ROW_LENGTH	
	RANGE	NULL	year(`orderDate`)	NULL	2006	64	256	1
	RANGE	NULL	year(`orderDate`)	NULL	2005	151	108	1
	RANGE	NULL	year(`orderDate`)	NULL	2004	111	147	1

Partitioning on Workbench



Comments:

☑ Enable Partitioning

Partition By:	RANGE	Parameters:	year(`orderDate`)	Partition Count:	5 ☑ Manu
Subpartition By:		Parameters:		Subpartition Count:	0 □ Manu

Partition	Values	Data Directory	Index Directory	Min Rows	Max Rows	Comment
part0	2004					
part1	2005	data_partition1				
part2	2006	data_partition2				
part3	2007	'data_partition3				
part4	2008	'data_partition4'				

MySQL Partitioning Limitations

- Foreign keys are not supported
- Partition tables do not support FULL TEXT searches
- All columns used in partitioning need to be part of every unique key in the table

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

Thanks you