



MYSQL Spatial v.s. PostGIS

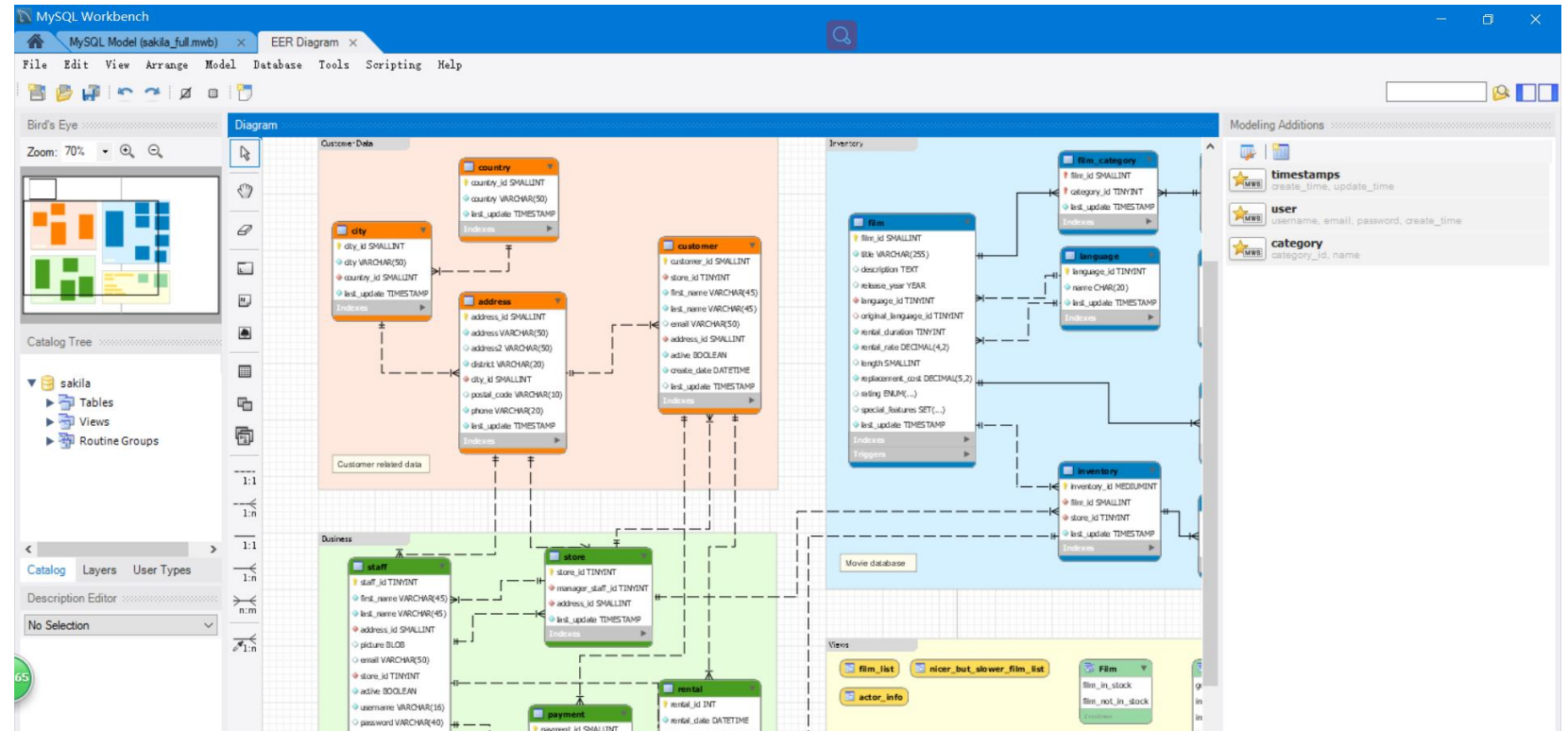
李岸洲

2018.6.4

GETTING STARTED



Navicat
for MySQL



BASIC DDL/DML:

- CREATE A TABLE:

```
1 mysql> CREATE TABLE pet (name VARCHAR(20), owner VARCHAR(20),  
2     -> species VARCHAR(20), sex CHAR(1), birth DATE, death DATE);
```

- LOAD DATA INTO A TABLE:

```
1 mysql> LOAD DATA LOCAL INFILE '/path/pet.txt' INTO TABLE pet  
2     -> LINES TERMINATED BY '\r\n';
```

```
1 mysql> INSERT INTO pet  
2     -> VALUES ('Puffball', 'Diane', 'hamster', 'f', '1999-03-30', NULL);
```

- USER-DEFINED VARIABLES:

Pass values from one statement to another.

Any type assignment.

```
1 mysql> SET @a='test';  
2 mysql> SELECT @a, (@a:=20) FROM tbl_name;
```



“GIS is a strategic and long term investment for MySQL”

- BACKGROUND:

Following the OGC specification, MySQL implements spatial extensions as a subset of the SQL with Geometry Types environment. This term refers to an SQL environment that has been extended with a set of geometry types. A geometry-valued SQL column is implemented as a column that has a geometry type.

Some spatial data types hold single geometry values:

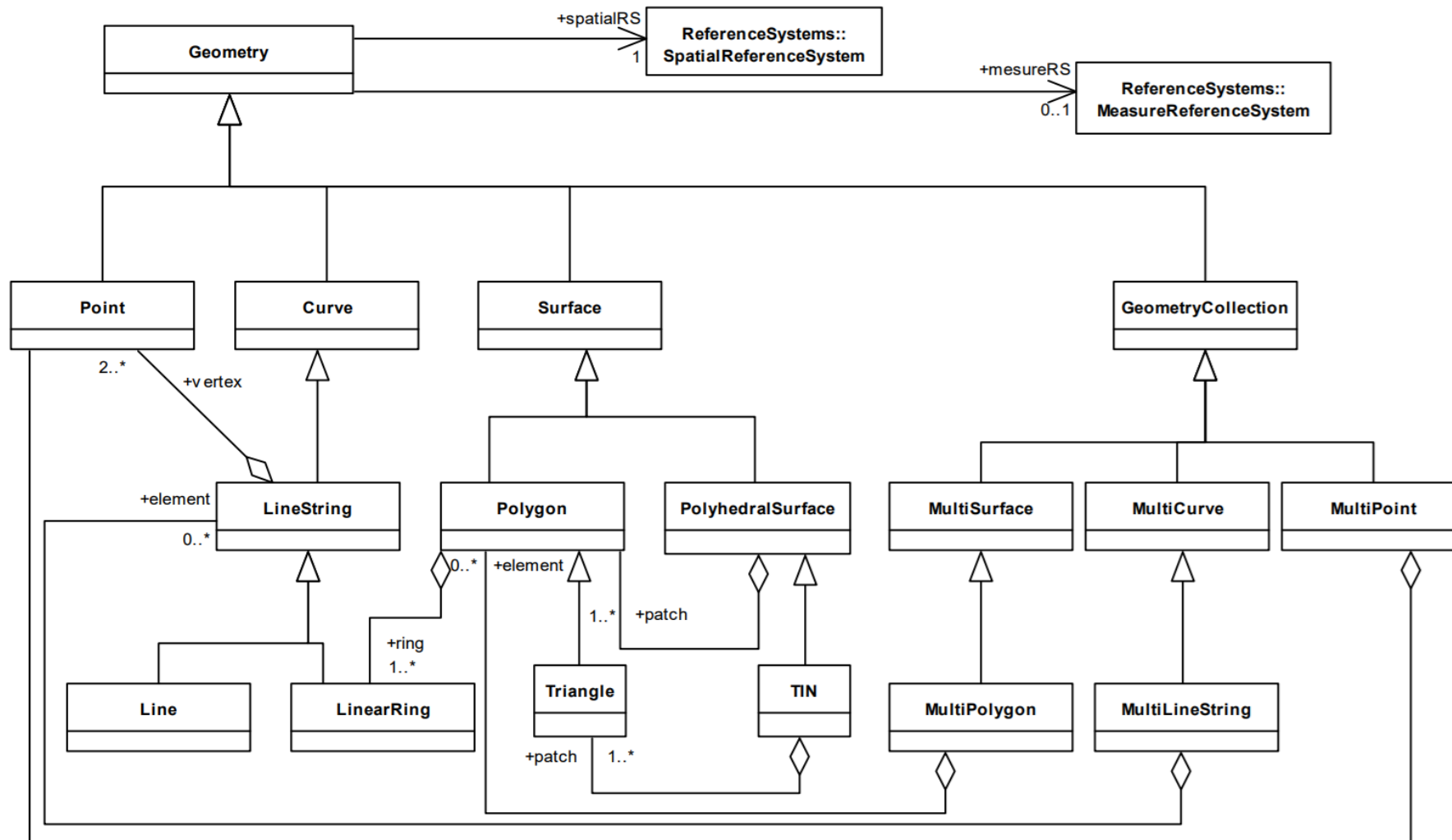
- GEOMETRY (store geometry values of any type)
- POINT
- LINESTRING
- POLYGON

The other spatial data types hold collections of values:

- MULTIPOINT
- MULTILINESTRING
- MULTIPOLYGON
- GEOMETRYCOLLECTION (store geometry values of any type) (NULL✓)



“GIS is a strategic and long term investment for MySQL”



“GIS is a strategic and long term investment for MySQL”

- Two standard spatial data formats are used to represent geometry objects in queries:
 - Well-Known Text (WKT) format
 - Well-Known Binary (WKB) format

Table WKB Components Example---POINT

Component ↴	Size ↴	Value ↴
Byte order ↴	1 byte ↴	01 ↴ little-endian
WKB type ↴	4 bytes ↴	01000000 ↴
X coordinate ↴	8 bytes ↴	000000000000F03F ↴
Y coordinate ↴	8 bytes ↴	000000000000F0BF ↴

using 4 bytes to indicate the SRID followed by the WKB representation of the value. [TOP]

```
CREATE TABLE geom (  
  p POINT NOT NULL SRID 0,  
  g GEOMETRY NOT NULL SRID 4326,  
  SPATIAL INDEX(g)  
);  
ALTER TABLE geom ADD pt POINT;  
ALTER TABLE geom DROP pt;
```

```
mysql> SELECT *  
FROM  
INFORMATION_SCHEMA.ST_SPATIAL_REFERENCE_SYSTEMS  
WHERE SRS_ID = 4326\G
```



“GIS is a strategic and long term investment for MySQL”

- Populating Spatial Columns

```
SET @g = 'LINESTRING(0 0,1 1,2 2)';  
INSERT INTO geom VALUES (ST_GeomFromText(@g));
```

```
SET @g = 'POLYGON((0 0,10 0,10 10,0 10,0 0),(5  
5,7 5,7 7,5 7, 5 5))';  
INSERT INTO geom VALUES (ST_GeomFromText(@g));
```

```
SET @g =  
'GEOMETRYCOLLECTION(POINT(1 1),LINESTRING(0 0,1  
1,2 2,3 3,4 4))';  
INSERT INTO geom VALUES (ST_GeomFromText(@g));
```

Syntactically well-formed and geometrically valid

```
ST_IsValid()
```



“GIS is a strategic and long term investment for MySQL”

- The most typical operations are:
 - Point queries that search for all objects that contain a given point;
 - Region queries that search for all objects that overlap a given region.
- MySQL uses R-Trees with quadratic splitting for SPATIAL indexes on spatial columns. A SPATIAL index is built using the minimum bounding rectangle (MBR) of a geometry.



R-TREE CONSTRUCTION:

ALGORITHM DESCRIPTION:

INSERTION:

CONDITION: nodes that overflow are split and propagate up the tree to insert E:

- 1 Find position for new record invoke **ChooseLeaf()** to select a leaf node L in which to place E;
- 2 If L has room for another entry, install E ; otherwise invoke **SplitNode()** to obtain L and LL containing E and all the old entries of L;
- 3 Invoke **AdjustTree()** on L, also passing LL if a split was performed [propagate change upwards];
- 4 If node split propagation caused the root to split, create a new root whose children are the 2 resulting nodes.

ChooseLeaf():

- 1 set N to be root
- 2 if N is a leaf, return
- 3 if N is not a leaf, let **F**l needs least enlargement to include E
- 4 set N to be the child node pointed to by **F**p and repeat from #2

Quadratic Split(): DIVID a set of M+1 INDEX entries into 2 groups [OVERFLOW]

1. **PickSeeds()**
2. If all entries have been assigned, return;
3. **PickNext()**, add it to the group whose covering rect have enlarged least, smaller area & fewer entries... repeat from #2.

PickSeeds():

find the pair with the most increasing area of new bbox to form GROUP1 & GROUP2

PickNext():

foreach not added, calculate the d1&d2(increasing area), return the max(|d1 -d2|)



Performance Analysis:

```
mysql> ALTER TABLE geom ADD SPATIAL INDEX(g);
```

```
Query OK, 32376 rows affected (4.05 sec)
```

```
Records: 32376 Duplicates: 0 Warnings: 0
```

```
mysql> SET @poly =
```

```
-> 'Polygon((30000 15000,
           31000 15000,
           31000 16000,
           30000 16000,
           30000 15000))';
```

```
mysql> SELECT fid,ST_AsText(g) FROM geom WHERE
-> MBRContains(ST_GeomFromText(@poly),g);
```

fid	ST_AsText(g)
21	LINESTRING(30000 15000, 31000 15000, 31000 16000, 30000 16000, 30000 15000) id: 1
22	LINESTRING(30000 15000, 31000 15000, 31000 16000, 30000 16000, 30000 15000) select_type: SIMPLE
23	LINESTRING(30000 15000, 31000 15000, 31000 16000, 30000 16000, 30000 15000) table: geom
24	LINESTRING(30000 15000, 31000 15000, 31000 16000, 30000 16000, 30000 15000) type: range
25	LINESTRING(30000 15000, 31000 15000, 31000 16000, 30000 16000, 30000 15000) possible_keys: g
26	LINESTRING(30000 15000, 31000 15000, 31000 16000, 30000 16000, 30000 15000) key: g
249	LINESTRING(30000 15000, 31000 15000, 31000 16000, 30000 16000, 30000 15000) key_len: 32
1	LINESTRING(30000 15000, 31000 15000, 31000 16000, 30000 16000, 30000 15000) ref: NULL
2	LINESTRING(30000 15000, 31000 15000, 31000 16000, 30000 16000, 30000 15000) rows: 50
3	LINESTRING(30000 15000, 31000 15000, 31000 16000, 30000 16000, 30000 15000) Extra: Using where
4	LINESTRING(30000 15000, 31000 15000, 31000 16000, 30000 16000, 30000 15000) 1 row in set (0.00 sec)
5	LINESTRING(30000 15000, 31000 15000, 31000 16000, 30000 16000, 30000 15000)
6	LINESTRING(30000 15000, 31000 15000, 31000 16000, 30000 16000, 30000 15000)
7	LINESTRING(30000 15000, 31000 15000, 31000 16000, 30000 16000, 30000 15000)
10	LINESTRING(30000 15000, 31000 15000, 31000 16000, 30000 16000, 30000 15000)
11	LINESTRING(30000 15000, 31000 15000, 31000 16000, 30000 16000, 30000 15000)
13	LINESTRING(30000 15000, 31000 15000, 31000 16000, 30000 16000, 30000 15000)
154	LINESTRING(30000 15000, 31000 15000, 31000 16000, 30000 16000, 30000 15000)
155	LINESTRING(30000 15000, 31000 15000, 31000 16000, 30000 16000, 30000 15000)
157	LINESTRING(30000 15000, 31000 15000, 31000 16000, 30000 16000, 30000 15000)

```
20 rows in set (0.00 sec)
```

SPATIAL indexes can be created on NOT NULL spatial columns and SRID attribute.

```
mysql> SET @poly =
```

```
-> 'Polygon((30000 15000,
           31000 15000,
           31000 16000,
           30000 16000,
           30000 15000))';
```

```
mysql> SELECT fid,ST_AsText(g) FROM geom IGNORE INDEX (g) WHERE
-> MBRContains(ST_GeomFromText(@poly),g);
```

fid	ST_AsText(g)
1	LINESTRING(30000 15000, 31000 15000, 31000 16000, 30000 16000, 30000 15000) id: 1
2	LINESTRING(30000 15000, 31000 15000, 31000 16000, 30000 16000, 30000 15000) select_type: SIMPLE
3	LINESTRING(30000 15000, 31000 15000, 31000 16000, 30000 16000, 30000 15000) table: geom
4	LINESTRING(30000 15000, 31000 15000, 31000 16000, 30000 16000, 30000 15000) type: ALL
5	LINESTRING(30000 15000, 31000 15000, 31000 16000, 30000 16000, 30000 15000) possible_keys: NULL
6	LINESTRING(30000 15000, 31000 15000, 31000 16000, 30000 16000, 30000 15000) key: NULL
7	LINESTRING(30000 15000, 31000 15000, 31000 16000, 30000 16000, 30000 15000) key_len: NULL
10	LINESTRING(30000 15000, 31000 15000, 31000 16000, 30000 16000, 30000 15000) ref: NULL
11	LINESTRING(30000 15000, 31000 15000, 31000 16000, 30000 16000, 30000 15000) rows: 32376
13	LINESTRING(30000 15000, 31000 15000, 31000 16000, 30000 16000, 30000 15000) Extra: Using where
21	LINESTRING(30000 15000, 31000 15000, 31000 16000, 30000 16000, 30000 15000) 1 row in set (0.00 sec)
22	LINESTRING(30000 15000, 31000 15000, 31000 16000, 30000 16000, 30000 15000)
23	LINESTRING(30000 15000, 31000 15000, 31000 16000, 30000 16000, 30000 15000)
24	LINESTRING(30000 15000, 31000 15000, 31000 16000, 30000 16000, 30000 15000)
25	LINESTRING(30000 15000, 31000 15000, 31000 16000, 30000 16000, 30000 15000)
26	LINESTRING(30000 15000, 31000 15000, 31000 16000, 30000 16000, 30000 15000)
154	LINESTRING(30000 15000, 31000 15000, 31000 16000, 30000 16000, 30000 15000)
155	LINESTRING(30000 15000, 31000 15000, 31000 16000, 30000 16000, 30000 15000)
157	LINESTRING(30000 15000, 31000 15000, 31000 16000, 30000 16000, 30000 15000)
249	LINESTRING(30000 15000, 31000 15000, 31000 16000, 30000 16000, 30000 15000)

```
20 rows in set (0.46 sec)
```



“GIS is a strategic and long term investment for MySQL”

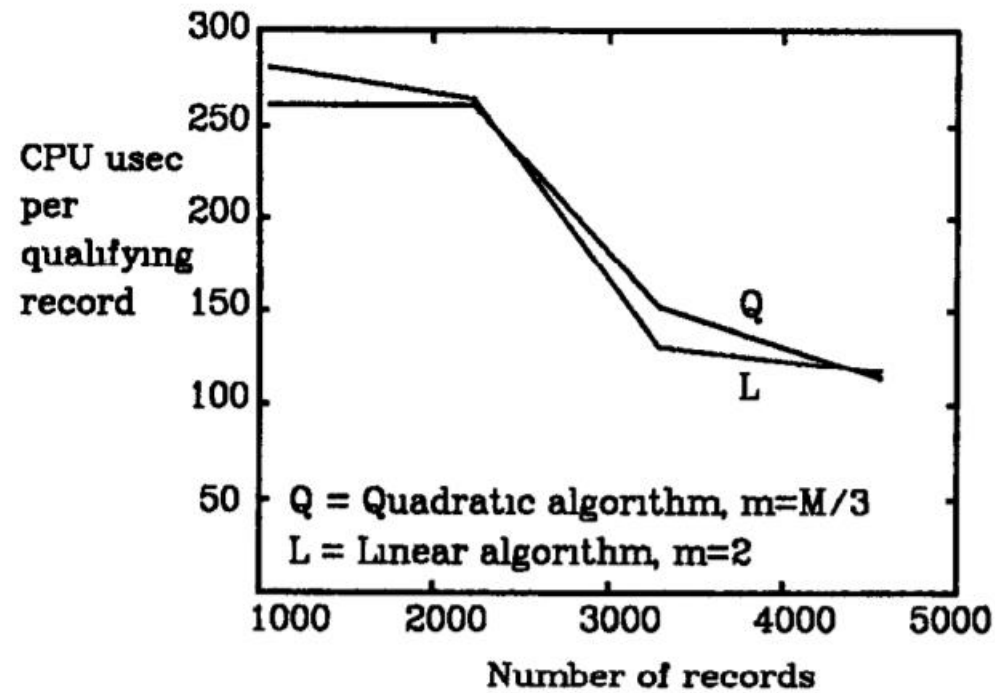


Figure 4 9
Search performance vs amount of data
CPU cost



CONCLUSION:

	MYSQL	POSTGIS
GEOMETRY TYPE	OGC's Geometry	Geography + geometry
DDL/DML on GEOMETRY COLUMN	Functions imported from Boost.geometry library mostly	Sphere + Cartesian Promise accuracy
SPATIAL INDEX	R-Tree	R-Tree based on GiST