# Running and illustrating queries on a Mysql Database

Di Huynh
University of Texas at Austin
di.huynh@cs.utexas.edu

Abstract - In this report, I ran and illustrated a set of queries on a Mysql Database. For the first part, I present an experiment where I assess the effectiveness of using indexes and figuring out the index that gives the best runtime. For the second part of this report, I use gnuplot to illustrate the result sets for these queries. The database I am using is a subset of the Data Release 8 from the Sloan Digital Sky Survey (SDSS).

## A. RUNNING THE QUERIES

### I. THE QUERIES

The database I am using is a subset of the Data Release 8 from the Sloan Digital Sky Survey (SDSS). I am working eight queries from the benchmark queries that SDSS provides in the DR7 and DR 8. They are mostly queries retrieving a range of tuples. The list of all of the queries is on the last page.

### II. THE DATABASE

The MYSQL database I'm using is on the compute-1-3 cluster, which is part of dbcluster. The database is called "test." The table is called sdss. Below is more information regarding the table.

| Table_name | Sdss |
|---|---|
| Version | 10 |
| Row_format | Fixed |
| Table_rows | 30,833,812 |
| Avg_row_length | 4075 |
| Data_length | 125,647,783,900 |
| Max_data_length | 1,147,010,530,095,923,199 |
| Index_length | 877,243,392 |
| Data_free | 0 |

### III. RUNTIME AND EXPLANATION

1. With objID (the primary key) as the only index.

| Id | Select_type | Table | Type | Poss keys | Key | Key_len | Ref | Rows | Extra |
|---|---|---|---|---|---|---|---|---|---|
| Q1 | Simple | G | All | Null | Null | Null | Null | 30833812 | Using where |
| Q2 | Simple | G | All | Null | Null | Null | Null | 30833812 | Using where |
| Q3 | Simple | G | All | Null | Null | Null | Null | 30833812 | Using where |

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | Simple | S | All | Null | Null | Null | Null | 30833812 | Using where |
| Q4 | Simple | G | All | Null | Null | Null | Null | 30833812 | Using where Using temporary Using filesort |
| Q5 | Simple | S | All | Null | Null | Null | Null | 30833812 | Using where |
| Q6 | Simple | S | All | Null | Null | Null | Null | 30833812 | Using where |
| Q7 | Simple | G | All | Null | Null | Null | Null | 30833812 | Using where |
| Q8 | Simple | G | All | Null | Null | Null | Null | 30833812 | Using where |

Runtime:

| | |
|---|---|
| Q1 | 16 mins 22.95 sec |
| Q2 | 18 mins 45.87 sec |
| Q3 | 16 mins 40.87 sec |
| Q4 | 22 mins 49.08 sec |
| Q5 | 15 mins 15.30 sec |
| Q6 | 16 mins 28.60 sec |
| Q7 | 16 mins 45.04 sec |
| Q8 | 16 mins 10.19 sec |
| AVG | 17 mins 17.9 sec |

With the size of the database being about 100 GB, the amount of time it takes to sequentially scan the entire database is about 15-18 mins. So the runtimes suggest that with the primary key as the only index, the database is scanning the entire table for each of the queries. This is confirmed furthermore by checking each queries with the EXPLAIN command. Q4 is slightly slower than the rest because the query has a GROUP BY clause, which requires extra sorting at the end.

Next I observe the frequency of standalone attributes and made 3 indexes on the most recurring ones.

| Standalone attributes | Frequency |
|---|---|
| r, decc, ra | 2 |
| (g, r), g, cModelMag_g, colv, parentID, rowv, extinction_r, (u,g,i), (colv, rowv), (u,g), (r,i), (i,z) | 1 |

- Blue text means that the query accesses g
- >1 hour counts as 60 mins in the total and average

2. With an index on ra.

| Table | Non_ unique | Key_name | Seq_in _index | Column _name | Collation | Cardinality | Index_type | Null |
|---|---|---|---|---|---|---|---|---|
| sdss | 1 | Index_ra | 1 | ra | A | 30833812 | BTREE | Yes |

| Queries | Not using the index | Using the index on ra |
|---|---|---|
| Q1 | 16 min 22.95 sec | 04 mins 47.50 secs |
| Q2 | 18 min 45.87 sec | 15 mins 35.10 secs |
| Q3 | 16 min 40.87 sec | 15 mins 45.69 secs |
| Q4 | 22 mins 49.08 sec | 22 mins 50.84 secs |
| Q5 | 15 mins 15.3 sec | 15 mins 45.12 secs |
| Q6 | 16 min 28.60 sec | 08 mins 46.00 secs |
| Q7 | 16 min 45.04 sec | 16 mins 34.45 secs |
| Q8 | 16 min 10.19 sec | 15 mins 35.32 secs |
| total | 136 mins 27.9 secs | 115 mins 40.02 secs |
| average | 16 mins 10.99 secs | 13 mins 42.5 secs |

The queries that use the index on ra:

| Q | Select_type | Table | Type | Poss keys | Key | Key_ Len | Rows | Extra |
|---|---|---|---|---|---|---|---|---|
| Q1 | Simple | S | Range | Index_ra | Index_ra | 9 | 132435 | Using where |
| Q6 | Simple | S | Range | Index_ra | Index_ra | 9 | 153254 | Using where |

The two queries with ra have greatly improved runtimes because the number of retrieved rows is much smaller than the total number of rows, so Mysql was able to use the index and access the qualified rows quickly.

3. With an index on r

| Table | Non_ unique | Key_name | Seq_in _index | Column _name | Collation | Cardinality | Index_type | Null |
|---|---|---|---|---|---|---|---|---|
| sdss | 1 | Index_r | 1 | R | A | 7709876 | BTREE | Yes |

I got the following runtime.

| Queries | Not using the index | index on r |
|---|---|---|
| Q1 | 16 min 22.95 sec | 21 mins 19.38 secs |
| Q2 | 18 min 45.87 sec | > 1 hour |
| Q3 | 16 min 40.87 sec | 16 mins 35.59 secs |
| Q4 | 22 mins 49.08 sec | > 1 hour |
| Q5 | 15 mins 15.3 sec | 19 mins 58.82 secs |

| | | |
|---|---|---|
| Q6 | 16 min 28.60 sec | 18 mins 37.45 secs |
| Q7 | 16 min 45.04 sec | 18 mins 01.43 secs |
| Q8 | 16 min 10.19 sec | 21 mins 11.35 secs |
| total | 136 mins 27.9 secs | 235 mins 44.02 secs |
| average | 16 mins 10.99 secs | 29 mins 20.5 secs |

The queries that use the index on r:

| Q | Select_type | Table | Type | Poss keys | Key | Key_ Len | Rows | Extra |
|---|---|---|---|---|---|---|---|---|
| Q2 | Simple | S | Range | Index_r | Index_r | 9 | 21382804 | Using where |
| Q4 | Simple | S | Range | Index_r | Index_r | 9 | 19557118 | Using where; using temporary; using filesort |

The number of rows is larger than 50% of the total number of rows. Since mysql takes a long time to finish the queries, I ran the queries both time using USE INDEX and IGNORE INDEX to see the difference. When USE INDEX is used, the runtime is over one hour. When IGNORE INDEX is used, I got the same results as when objID is the only index. This suggests that mysql is using the index to retrieve the qualified tuples. Mysql is making a significant number of random reads by using the index. Since there are so many qualified tuples, a full table scan would be much faster as it can just read from disk sequentially.

4. With an index on decc

| Table | Non_ unique | Key_name | Seq_in _index | Column _name | Collation | Cardinality | Index_type | Null |
|---|---|---|---|---|---|---|---|---|
| sdss | 1 | Index_r | 1 | R | A | 30833812 | BTREE | Yes |

| Queries | Not using the index | index on decc |
|---|---|---|
| Q1 | 16 min 22.95 sec | 04 mins 21.28 secs |
| Q2 | 18 min 45.87 sec | 18 mins 35.75 secs |
| Q3 | 16 min 40.87 sec | 15 mins 24.49 secs |
| Q4 | 22 mins 49.08 sec | 22 mins 32.22 secs |
| Q5 | 15 mins 15.3 sec | 16 mins 43.56 secs |
| Q6 | 16 min 28.60 sec | > 1 hour |
| Q7 | 16 min 45.04 sec | 21 mins 16.11 secs |
| Q8 | 16 min 10.19 sec | 16 mins 04.53 secs |
| total | 136 mins 27.9 secs | 174 mins 57.94 secs |
| Average | 16 mins 10.99 secs | 21 mins 22.24 secs |

The runtime for Q1 has significantly improved as a result of the index. Q6, however, is much slower even though the two queries are similar. So I check the explanation for those two queries.

| Q | Select_type | Table | Type | Poss keys | Key | Key_Len | Rows | Extra |
|---|---|---|---|---|---|---|---|---|
| Q1 | Simple | S | range | Index_decc | Index_decc | 9 | 1628867 | Using where |
| Q6 | Simple | S | range | Index_decc | Index_decc | 9 | 22866151 | Using where |

Since Q1 has a much smaller result set, mysql is using the index correctly and gives a good runtime. Q6's result set however is much larger, so by using the index, mysql is making too many random reads.

## III. SUGGESTIONS

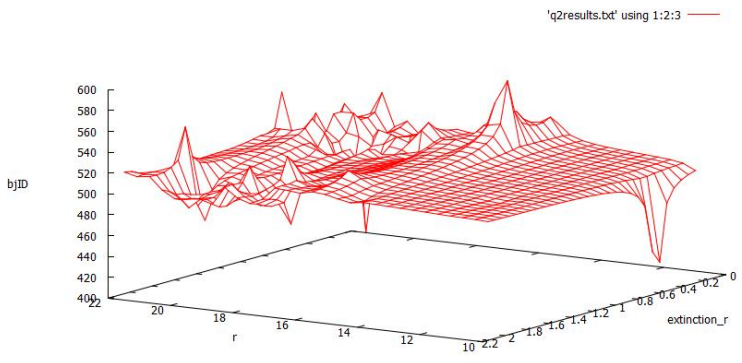
From the experiments above, I conclude that Mysql indexes only works well when the number of qualified tuples is much smaller than the total number of rows, around 30%. When the result set exceeds that number, if the index is used, the runtime will be significantly slower than a full table scan. A possible explanation is that when Mysql uses the index, the database needs one seek to find the tuple, then another one to find it on disk. When the resulting set of tuples is large, the I/O's add up and slow down the query considerably. With a table scan, the database only needs to find the first tuple in the range and can just sequentially read from disk, eliminating the need for the cost for extra seeks.

So for this set of queries and data, an index on ra gives the best performance in both time and space since the queries involving ra are quite selective, which leads to effective use of the index.

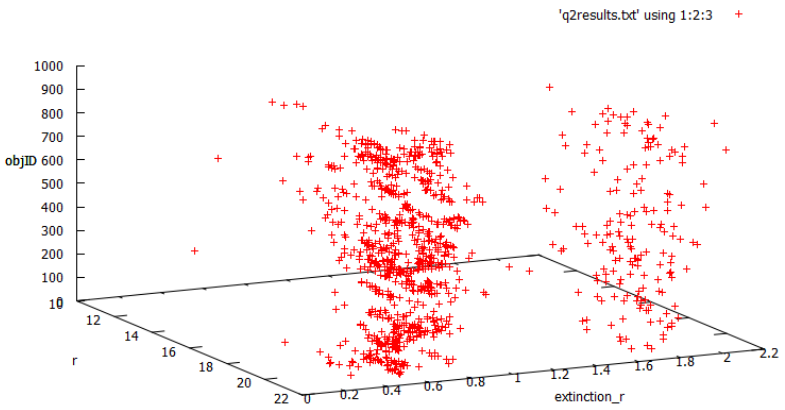
## B. ILLUSTRATING THE QUERIES

Since Q1 has an empty result set, I only plotted 7 queries.

Q2:



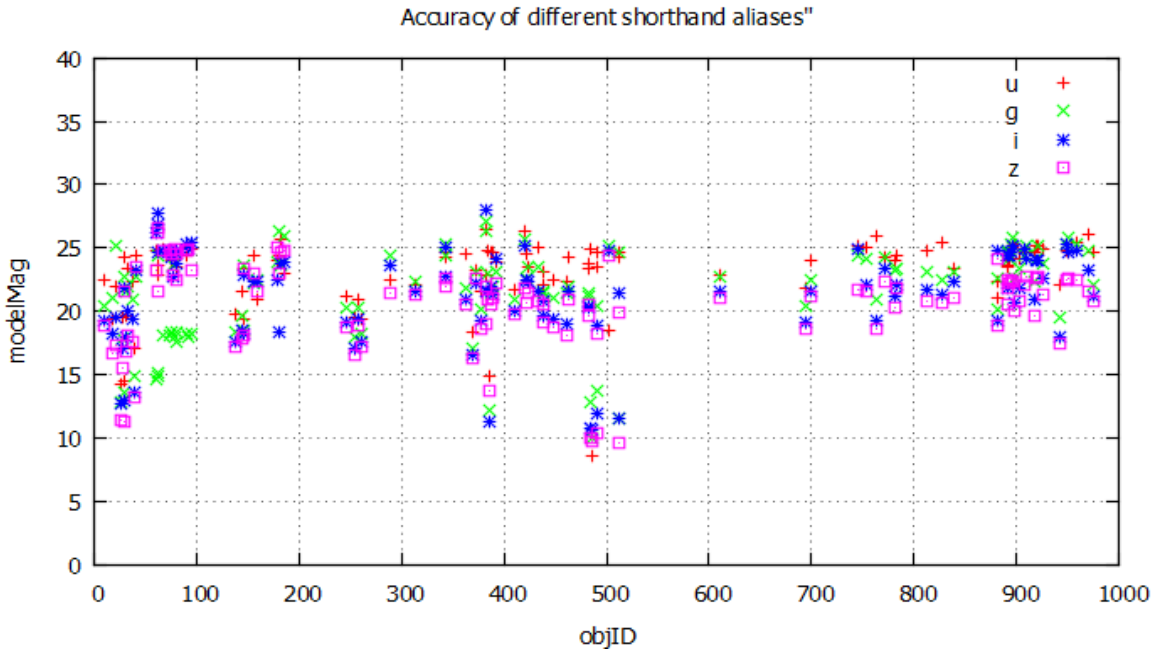

```
Script:

Set autoscale

Set dgrid3d 30,30

Set hidden3d

Splot 'q2results.txt'

with lines
```




Script:

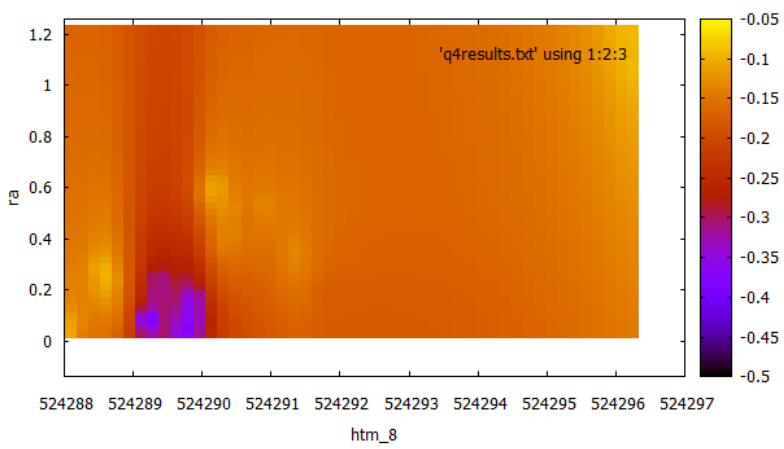
```
set autoscale

set xlabel

“extinction_r”

set ylabel “objid”

splot ‘q2results.txt’
```

Q3:



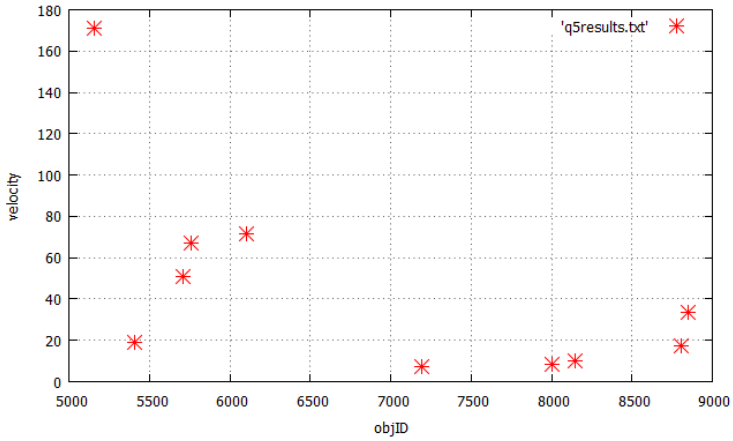

```
Script:
Set autoscale
Set xlabel ‘objID’
Set ylabel ‘modelMag’
Set title “Accuracy of different shorthand aliases”
Set yrange [0:40]
Set grid
Plot ‘q3results’ using 1:2 title ‘u’ with points ls 1,\
‘q3results’ using 1:3 title ‘g’ with points ls 2,\
‘q3results’ using 1:4 title ‘i’ with points ls 3,\
‘q3results’ using 1:5 title ‘z’ with points ls 4
```

Q4:



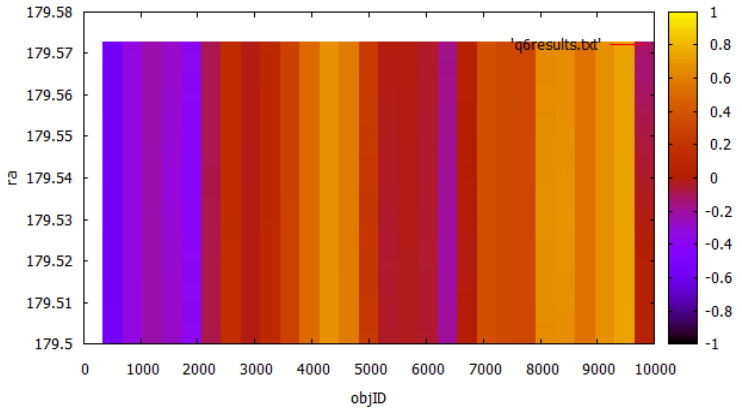

```
Script:

set pm3d map
set palette
set dgrid3d 30,30
set hidden3d
set xlabel 'htm_8'
set ylabel 'ra'
splot 'q4results.txt'
using 1:2:3
```

Q5:



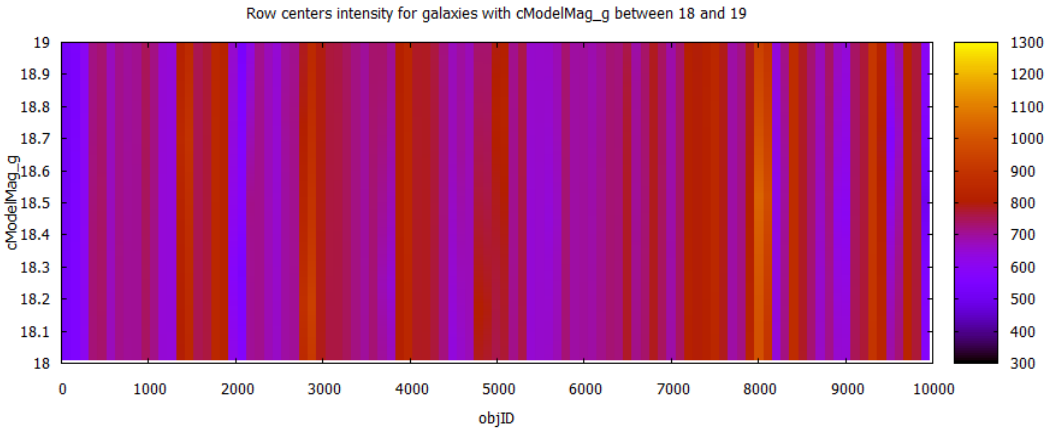

```
Script:

set grid
set xlabel "objid"
set ylabel "velocity"
plot 'q5results.txt'
pt3 ps 2
```

Q6



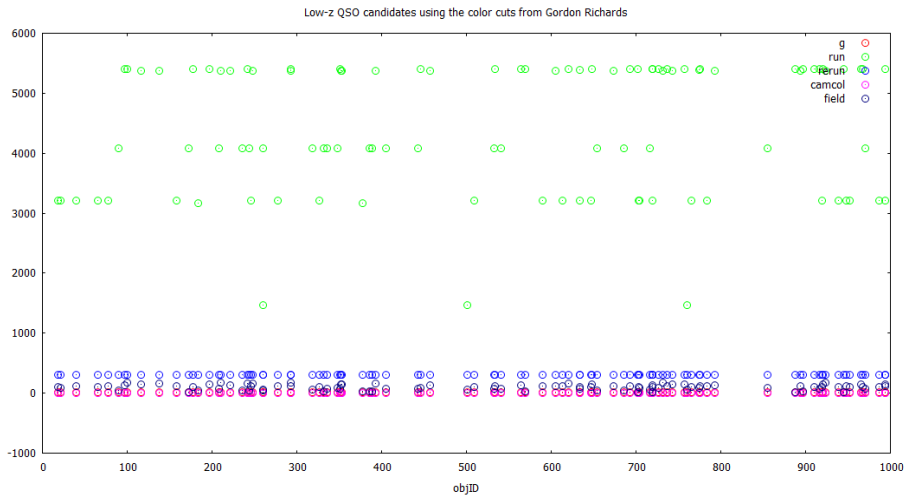

```
script:

set autoscale
set pm3d map
set dgrid3d 30,30
set xlabel 'objid'
set ylabel 'ra'
splot 'q6results.txt' u
1:2:3
```

Q7:




```
Script:
set autoscale
set pm3d map
set dgrid3d 100,100
set xlabel 'objid'
set ylabel 'cmodelmag_g'
splot 'q6results.txt' u 1:2:3
```

Q8:




```
Script:
set autoscale
set xlabel 'objID'
set title "Low-z QSO candidates using the color cuts from
Gordon Richards"
set yrange [-1000:6000]
plot 'q8results' using 1:2 title 'g' with pt 6,\
'q8results' using 1:3 title 'run' with points pt 6,\
'q8results' using 1:4 title 'rerun' with points pt 6,\
'q8results' using 1:5 title 'camcol' with points pt 6,\
'q8results' using 1:6 title 'field' with points pt6
```

# QUERIES

Q1. Find all galaxies with blue surface brightness between 23 and 25 mag per square arcseconds, and -10 < supergalactic latitude (sgb) < 10, and declination less than zero.

```
SELECT G.objID
FROM Galaxy G
WHERE G.ra > 250 AND G.ra < 270
AND G.decc > 50
AND (G.g+ (5*log(G.r)) > 23) and (G.g+(5*log(G.r)))<25;
```

Q2. Find all galaxies brighter than magnitude 22, where the local extinction is >0.75.

```
SELECT G.objID
FROM Galaxy G
WHERE G.r<22 AND G.extinction_r>0.175;
```

Q3. Find galaxies that are blended with a star, output the deblended magnitudes.

```
SELECT G.objID, G.u, G.g, G.r, G.i, G.z
FROM Galaxy G, Star S
WHERE G.parentID >0 AND S.parentID = G.parentID;
```

Q4. Create a count of galaxies for each of the HTM triangles (hierarchal triangular mesh) which satisfy a certain color cut, like 0.7u-0.5g-0.2 and i-magr<1.25 and r-mag<21.75, output it in a form adequate for visualization.

```
SELECT (G.htmID/POWER(2,24)) AS htm_8,
       AVG(G.ra) AS ra, AVG(G.decc) AS decc,
       COUNT(*) AS pop
FROM Galaxy G
WHERE (0.7*G.u - 0.5*G.g - 0.2*G.i) < 1.25
     AND (G.r <21.75)
GROUP BY (G.htmID/POWER(2,24));
```

Q5. Provide a list of moving objects consistent with an asteroid.

```
SELECT S.objID,
SQRT(POWER(S.rowv,2) + POWER(S.colv, 2)) AS velocity
FROM sdss S
WHERE (POWER(S.rowv,2) + POWER(S.colv, 2)) > 50
     AND S.rowv >= 0 AND S.colv >=0;
```

Q6. The rectangular search using straight coordinate constraints

```
SELECT S.objID, S.ra, S.decc
FROM sdss S
WHERE (S.ra between 179.5 and 182.3) and (S.decc between -1.0 and 1.8);
```

Q6. Find galaxies with g magnitudes between 18 and 19

```
SELECT G.objID, G.cModelMag_g
FROM Galaxy G
WHERE G.cModelMag_g BETWEEN 18 and 19;
```

Q7. Find the top 100 low-z QSO candidates using the color cuts from Gordon Richards.

```
SELECT G.g, G.run, G.rerun, G.camcol, G.field, G.objID
        FROM Galaxy G
        WHERE ((G.g <= 22) AND (G.u - G.g >= -0.27)
AND (G.u - G.g < 0.71) AND (G.g - G.r >= -0.24)
AND (G.g - G.r < 0.35) AND (G.r - G.i >= -0.27)
AND (G.r - G.i < 0.57) AND (G.i - G.z >= -0.35)
AND (G.i - G.z < 0.70));
```