Patrick Trinkle
CMSC661
Spring 2008
HW2

How MySQL chooses to handle indices.

MySQL supports three data structures for indices: B-trees; R-tress; and hash tables. The type of index needed and the type of data involved determine the type of valid index. To handle the actual creation and usage of indices MySQL employs storage engines. B-Trees are used to handle non-spatial data. More specifically PRIMARY KEY, UNIQUE, INDEX AND FULLTEXT indices are handled with B-Trees. R-Trees handle all spatial data. The values for all indices are formed by concatenating user specified columns. MySQL uses an optimizer to determine the value of using an index to locate rows or whether it would be more beneficial to scan the data blocks. This is such that if a query is likely to retrieve many rows it is faster to do a file scan as this will reduce the seek time for block reads.

The syntax for creating an index is as follows:

```
CREATE [UNIQUE|FULLTEXT|SPATIAL] INDEX index_name
[index_type] ON tbl_name (index_col_name,…)

index_col_name:
     col_name [(length)] [ASC | DESC]

index_type:
     USING {BTREE | HASH | RTREE}
```

MySQL will ignore the Ascending or Descending specified on the index_col_name. This is for future compatibility as currently only ascending indexes are supported. The length field is relevant if your data is unique on the first part of the data value; i.e. the first few digits of a shipping label or something. Also if single-column indexes exist on columns in a query the optimizer may attempt to merge them. If this is not possible an alternative is to use the index most likely to return the least rows; as this will shortcut the other part of the query's dataset.

MySQL 5.0 Reference Manual "12.1.4 CREATE INDEX Syntax"
http://dev.mysql.com/doc/refman/5.0/en/create-index.html

MySQL 5.0 Reference Manual "How MySQL Uses Indexes"
http://dev.mysql.com/doc/refman/5.0/en/mysql-indexes.html