Optimizing MySQL Usage

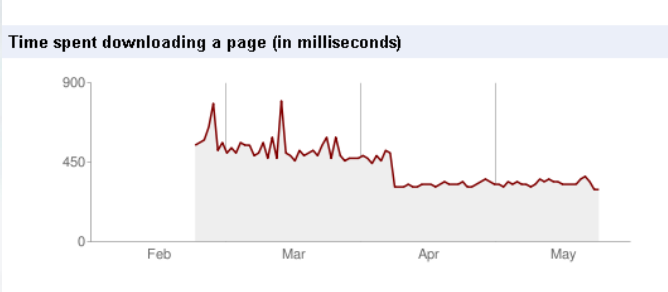
Rene Churchill
WherezIt.com &
AstuteComputing.com

rene@astutecomputing.com

My goal here is to throw out enough different ideas so that everybody hears at least one or two things that may be useful for them.

Where are the bottlenecks?

First Step: Figure out where the problems are...



You can't improve what you can't measure.

Basis of ISO 9000 improvement process.

Go for the low hanging fruit but do it intelligently. If the slow query is only run as part of the monthly statistics, who cares? Speed up the queries that the public sees first.

/etc/my.cnf

```
[mysqld]
log-slow-queries = slow.log
long_query_time = 1
log-queries-not-using-indexes
```

Prior to MySQL 5.1.21, the minimum value of long_query_time is 1, and the value for this variable must be an integer.

Beginning with MySQL 5.1.21, the minimum is 0, and a resolution of microseconds is supported when logging to a file. However, the microseconds part is ignored and only integer values are written when logging to tables.

Old / Slow Hardware

Don't have to upgrade MySQL
Use slower hardware.

My development server:
(recently retired) 500Mhz P3 Pentium

Sometimes you can't modify the customer's environment and upgrade MySQL just to make your life easier.

Replicate their environment on your development server. Use an old server to slow things down.

Counting Queries

10,000 fast queries are worse than 1 slower one

Reducing the query count generally requires refactoring / rewriting the function that is doing all of the queries.

Outside the scope of this talk

header.php

```
if (!$Timer_Start) {
    list($Timer_uStart, $Timer_Start) = explode(" ",microtime());
}

$QUERY_COUNT = 0;
$QUERY_TIME  = 0;
$DEBUG_MYSQL = 0;
```

mysql_class.php

```
function Query ($query) {
    global $QUERY_COUNT, $QUERY_TIME, $DEBUG_MYSQL;
    $QUERY_COUNT++;

    // Start the timer so we can track query times
    list($Timer_uStart, $Timer_Start) = explode(" ",microtime());

    $this->result = @mysql_query($query, $this->id) or
        MySQL_ErrorMsg ("Unable to perform query: $query");

    list($Timer_uEnd, $Timer_End) = explode(" ",microtime());
    $QUERY_TIME += ($Timer_End+$Timer_uEnd) - ($Timer_Start+$Timer_uStart);
    if ($DEBUG_MYSQL) {
        $tmp = ($Timer_End+$Timer_uEnd) - ($Timer_Start+$Timer_uStart);
        $elapsed = sprintf("%.3f",$tmp);
        print("\n\n<pre>\n$query\nElapsed Time: $elapsed\n\n");
        $stack_trace = debug_backtrace();
        foreach ($stack_trace as $tmp) {
            print"Function: ".$tmp['function']."\nLine: ".
                $tmp['line']."\nFile: ".$tmp['file']."\n";
        }
        print("</pre>\n");
    }
}
```

footer.php

```
global $Timer_Start, $Timer_uStart, $QUERY_COUNT, $QUERY_TIME;

list($Timer_uEnd, $Timer_End) = explode(" ",microtime());
$tmp = ($Timer_End + $Timer_uEnd) - ($Timer_Start +
$Timer_uStart);
$elapsed = sprintf("%.3f",$tmp);
print("<p class=\"gray_text\">Page generated in: $elapsed
seconds\n");
$query_sec = sprintf("%.3f", $QUERY_TIME);
print("<br>Query Count: $QUERY_COUNT Query Time: $query_sec\n");
```

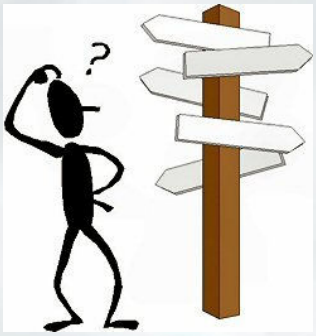
Pick a query to optimize

```
  SELECT DISTINCT l.listing_id,
         IF(l.created_on > DATE_SUB(NOW(), INTERVAL 3 DAY),1,0) AS NewAd
    FROM wherezit_cache.listing_cache_ac0a1d5d9c321268da6de781fdcf90 AS lc,
         Listings as l,
         Listing_photo_xref AS lpx,
         Category_listing_address_xref AS clax
   WHERE lc.listing_id = l.listing_id
     AND l.listing_type IN ('classified','classified_home',
                            'classified_vehicle','classified_job')
     AND l.status = 'Ok'
     AND l.valid_thru >= NOW()
     AND lpx.listing_id = l.listing_id
     AND l.listing_id = clax.listing_id
ORDER BY NewAd DESC, RAND()
   LIMIT 150
Elapsed Time: 0.823

Function: Query
Line: 1830
File: /home/wherezit/www/html/lib/listing_classified_class.php
Function: DisplayRecentRandomAds
Line: 46
File: /home/wherezit/www/html/index.php
```

What the heck is happening?

Second step: Figure out what is going on....



EXPLAIN is your friend

Running EXPLAIN on a query shows how MySQL is deciding which indexes to use in a SELECT query.

http://dev.mysql.com/doc/refman/5.1/en/using-explain.html

http://www.mysqlperformanceblog.com/

```
[wherezit]> EXPLAIN SELECT listing_id
                      FROM Listings
                     WHERE created_on > 20090501
                       AND listing_type = 'news'
                       AND listing_id > 300000 \G

******************** 1. row ********************
           id: 1
  select_type: SIMPLE
        table: Listings
         type: range
possible_keys: PRIMARY,type_created
          key: type_created
      key_len: 11
          ref: NULL
         rows: 1020
        Extra: Using where
1 row in set (0.00 sec)
```

\G displays the results vertically instead of horizontally

Fixing the Problem

Last step is to resolve the problem

Test & Measure

Try, try, try again

Basic Indexes

Creating Indexes - First step towards performance

Index columns in your WHERE clause first

DO NOT index everything - Read / Write trade-off

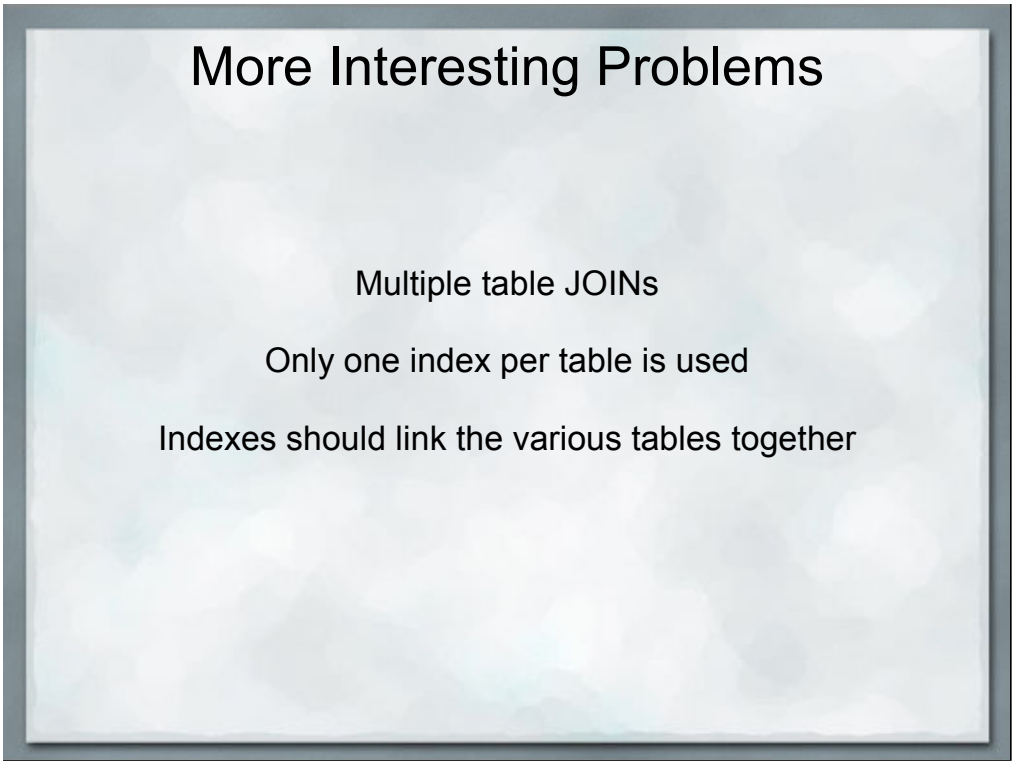
String Indexes

Beware LIKE clauses

Index can help WHERE name LIKE 'John%'

Cannot help 'WHERE name LIKE '%reggie%'
Avoid these queries if possible

Use the ENUM column type
instead of CHAR or VARCHAR



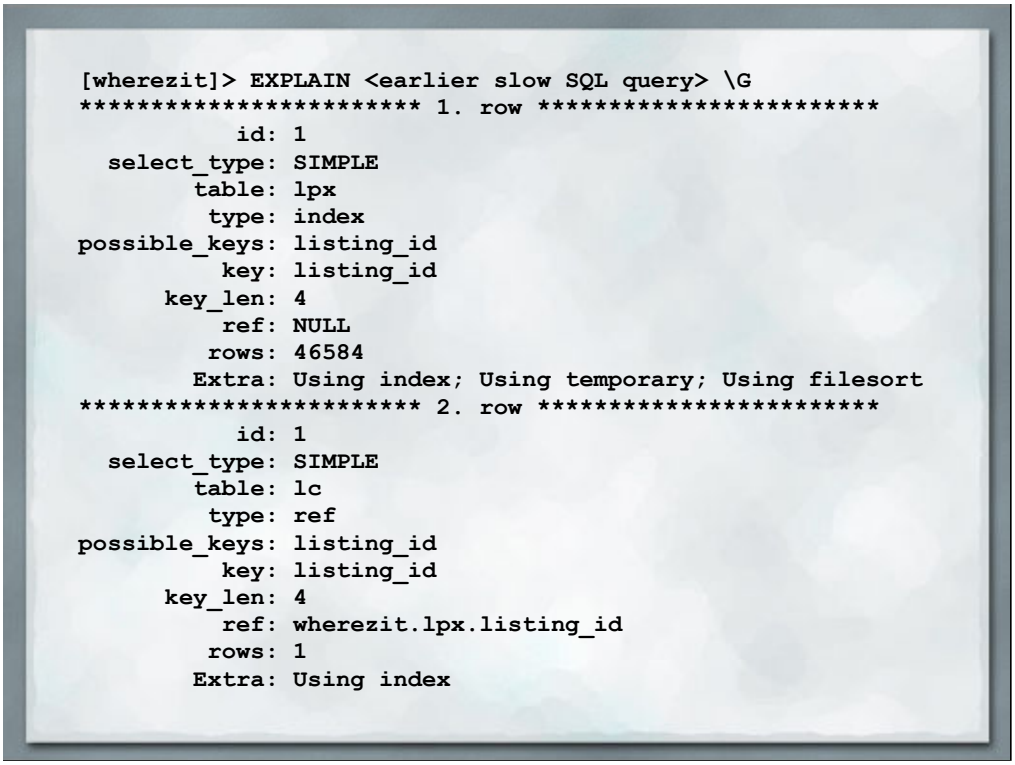
Relational databases are all about linking the data in multiple tables together, it's the whole point, otherwise we'd all be using a single large flat-file to store all of our data.

EXPLAIN will show the order in which the tables are joined together.

A More Interesting Query

```
  SELECT DISTINCT l.listing_id,
         IF(l.created_on > DATE_SUB(NOW(), INTERVAL 3 DAY),1,0) AS NewAd
    FROM wherezit_cache.listing_cache_ac0a1d5d9c321268da6de781fdcf90 AS lc,
         Listings as l,
         Listing_photo_xref AS lpx,
         Category_listing_address_xref AS clax
   WHERE lc.listing_id = l.listing_id
     AND l.listing_type IN ('classified','classified_home',
                            'classified_vehicle','classified_job')
     AND l.status = 'Ok'
     AND l.valid_thru >= NOW()
     AND lpx.listing_id = l.listing_id
     AND l.listing_id = clax.listing_id
ORDER BY NewAd DESC, RAND()
   LIMIT 150
Elapsed Time: 0.823

Function: Query
Line: 1830
File: /home/wherezit/www/html/lib/listing_classified_class.php
Function: DisplayRecentRandomAds
Line: 46
File: /home/wherezit/www/html/index.php
```



```
[wherezit]> EXPLAIN <earlier slow SQL query> \G
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: lpx
         type: index
possible_keys: listing_id
          key: listing_id
      key_len: 4
          ref: NULL
         rows: 46584
        Extra: Using index; Using temporary; Using filesort
*************************** 2. row ***************************
           id: 1
  select_type: SIMPLE
        table: lc
         type: ref
possible_keys: listing_id
          key: listing_id
      key_len: 4
          ref: wherezit.lpx.listing_id
         rows: 1
        Extra: Using index
```

MySQL joins tables in order. Generally selecting the table that returns the fewest rows first.

Then linking the appropriate next table based on the WHERE clause and how the tables are joined together.

Limitations of EXPLAIN syntax, doesn't always get the order of the tables correct.

```
*************************** 3. row ***************************
           id: 1
  select_type: SIMPLE
        table: l
         type: eq_ref
possible_keys: PRIMARY,type_created
          key: PRIMARY
      key_len: 4
          ref: wherezit_cache.lc.listing_id
         rows: 1
        Extra: Using where
*************************** 4. row ***************************
           id: 1
  select_type: SIMPLE
        table: clax
         type: ref
possible_keys: listing_id
          key: listing_id
      key_len: 4
          ref: wherezit_cache.lc.listing_id
         rows: 1
        Extra: Using where; Using index; Distinct
4 rows in set (0.01 sec)
```

Avoid Unecessary Joins

```
  SELECT DISTINCT l.listing_id,
         IF(l.created_on > DATE_SUB(NOW(), INTERVAL 3 DAY),1,0) AS NewAd
    FROM wherezit_cache.listing_cache_ac0a1d5d9c321268da6de781fdcf90 AS lc,
         Listings as l,
         Listing_photo_xref AS lpx,
         Category_listing_address_xref AS clax
   WHERE lc.listing_id = l.listing_id
     AND l.listing_type IN ('classified','classified_home',
                            'classified_vehicle','classified_job')
     AND l.status = 'Ok'
     AND l.valid_thru >= NOW()
     AND lpx.listing_id = l.listing_id
     AND l.listing_id = clax.listing_id
ORDER BY NewAd DESC, RAND()
   LIMIT 150
```

Look out for leftovers from previous development.

In this query the clax table is unecessary, no columns from it are referenced by any other table or in the SELECT clause.

Avoid Unecessary Joins

```
  SELECT DISTINCT l.listing_id,
         IF(l.created_on > DATE_SUB(NOW(), INTERVAL 3 DAY),1,0) AS NewAd
    FROM wherezit_cache.listing_cache_ac0a1d5d9c321268da6de781fdcf90 AS lc,
         Listings as l,
         Listing_photo_xref AS lpx,
         Category_listing_address_xref AS clax
   WHERE lc.listing_id = l.listing_id
     AND l.listing_type IN ('classified','classified_home',
                            'classified_vehicle','classified_job')
     AND l.status = 'Ok'
     AND l.valid_thru >= NOW()
     AND lpx.listing_id = l.listing_id
     AND l.listing_id = clax.listing_id
ORDER BY NewAd DESC, RAND()
   LIMIT 150
```

Avoid use of tmp tables

```
[wherezit]> EXPLAIN <earlier slow SQL query> \G
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: lpx
         type: index
possible_keys: listing_id
          key: listing_id
      key_len: 4
          ref: NULL
         rows: 46584
        Extra: Using index; Using temporary; Using filesort
```

Lots of disk I/O is the problem with this.

Generally avoid large sort operations in SQL, these often trigger the use of tmp tables

However, sometimes MySQL can sort the results faster than PHP can.

ORDER BY can cause hassles.

At least move tmp tables into memory

```
[wherezit]> show variables like 'tmp_table%';
+----------------+----------+
| Variable_name  | Value    |
+----------------+----------+
| tmp_table_size | 33554432 |
+----------------+----------+
```

/etc/my.cnf

```
[mysqld]
tmp_table_size = 64Mb
max_heap_table_size = 64Mb
```

tmp_table_size default is 32Mb

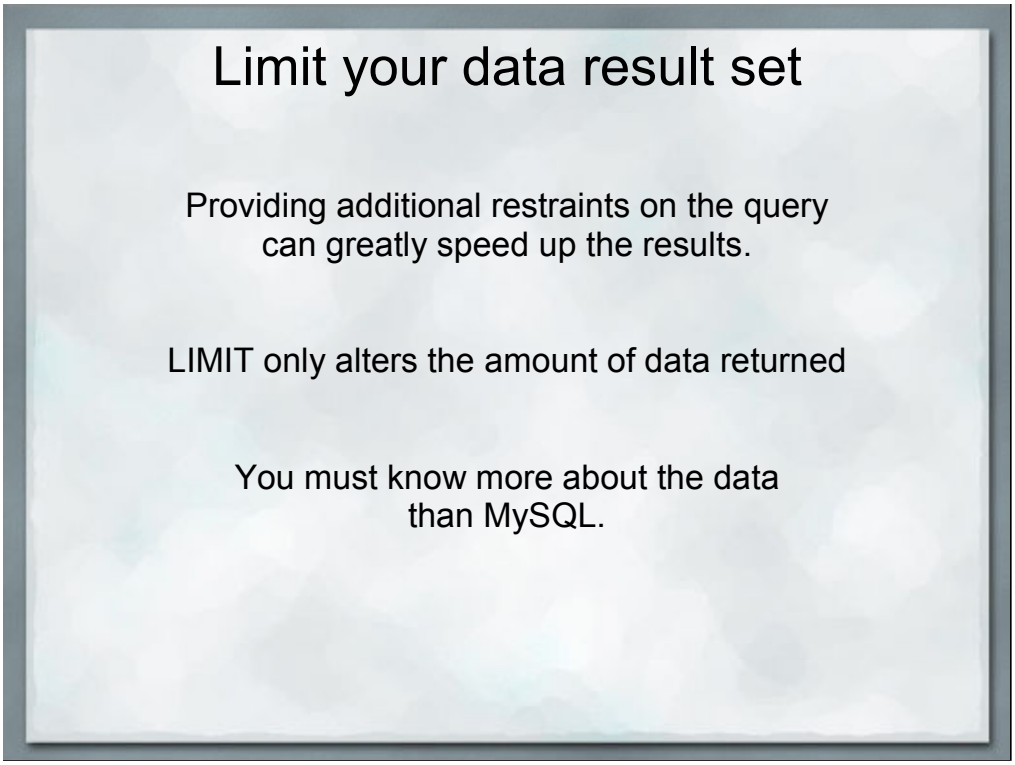
Need to check size of max_heap_table_size too, since MySQL uses the smaller of the two values.

Forcing MySQL to use a specific index

Generally avoid doing this
MySQL is pretty good about choosing the right index

```
SELECT ...
  FROM table1 FORCE (index1, index2,...)

SELECT ...
  FROM table1 IGNORE (index1, index2,...)
```



LIMIT only affects the amount of data returned from the query, not the amount of information that MySQL has to handle.

Limit your data result set

```
  SELECT DISTINCT l.listing_id,
         IF(l.created_on > DATE_SUB(NOW(), INTERVAL 3 DAY),1,0) AS NewAd
    FROM wherezit_cache.listing_cache_ac0a1d5d9c321268da6de781fdcf90 AS lc,
         Listings as l,
         Listing_photo_xref AS lpx
   WHERE lc.listing_id = l.listing_id
     AND l.listing_type IN ('classified','classified_home',
                            'classified_vehicle','classified_job')
     AND l.status = 'Ok'
     AND l.valid_thru >= NOW()
     AND l.created_on > DATE_SUB(NOW(), INTERVAL 30 DAY)
     AND lpx.listing_id = l.listing_id
ORDER BY NewAd DESC, RAND()
   LIMIT 150
```

```
Elapsed Time: 0.011
```

```
[wherezit]> EXPLAIN <query> \G
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: l
         type: range
possible_keys: PRIMARY,type_created
          key: type_created
      key_len: 11
          ref: NULL
         rows: 1103
        Extra: Using where; Using temporary; Using filesort
*************************** 2. row ***************************
           id: 1
  select_type: SIMPLE
        table: lc
         type: ref
possible_keys: listing_id
          key: listing_id
      key_len: 4
          ref: wherezit.l.listing_id
         rows: 1
        Extra: Using index; Distinct
*************************** 3. row ***************************
           id: 1
  select_type: SIMPLE
        table: lpx
         type: ref
possible_keys: listing_id
          key: listing_id
      key_len: 4
          ref: wherezit_cache.lc.listing_id
         rows: 3
        Extra: Using where; Using index; Distinct
```

Note that this query is still using the temporary table & filesort. The filesort is caused by the ORDER BY clause but it is faster to let MySQL handle that.

Always test your assumptions

The big change is the number of rows that are selected from the Listings table - 1103, a much smaller set of data to work with, thus quicly linked to the other tables and sorted.

Other tricks to try

The tips that follow work occasionally,
try them AFTER you have already fidded with indexes

Generally because the gains you can get from some of these tips is smaller than correctly applying indexes.

Sorting outside of MySQL

Occasionally useful, will often remove use of tmp tables

See PHP's usort() function

Advanced Indexes

SELECT will answer queries from Indexes alone if possible, very fast
beware read/write trade-off if adding extra indexes

```
SELECT COUNT(listing_id)
  FROM Listings
 WHERE listing_id > 100000
   AND listing_id < 300000
```

Avoid use of DISTINCT

Avoid the use of DISTINCT, forces MySQL to do a sort of the results, often unnecessarily

Use PHP associative arrays instead, sometimes faster.

```
$max = mysql_num_rows($rslt);
$tmp = array();
for ($i = 0; $i < $max; $i++) {
    list($a) = mysql_fetch_row($rslt);
    $tmp[$a]++;
}
$distinct_array = array_keys($data);
```

If you need to throw DISTINCT into the query and you're not sure why, that's a red flag that you've written the query incorrectly.

The main thing we're hoping to reduce here is the need for MySQL to suffer the file I/O of writing the temp file to disk and then sorting it.

Be careful using IN

```
SELECT *
  FROM Listings
 WHERE listing_id IN (1,3,5,7,.....)
```

```
$query = "SELECT *
            FROM Listings
           WHERE listing_id IN (".
          join(',',$listing_id_array).")";
```

What happens when $listing_id_array has 100,000 values?

Sometimes the CPU time required to just PARSE the SQL query becomes signifigant.

This can often bite you when transitioning from test data to real-world data sets, which tend to be MUCH larger.

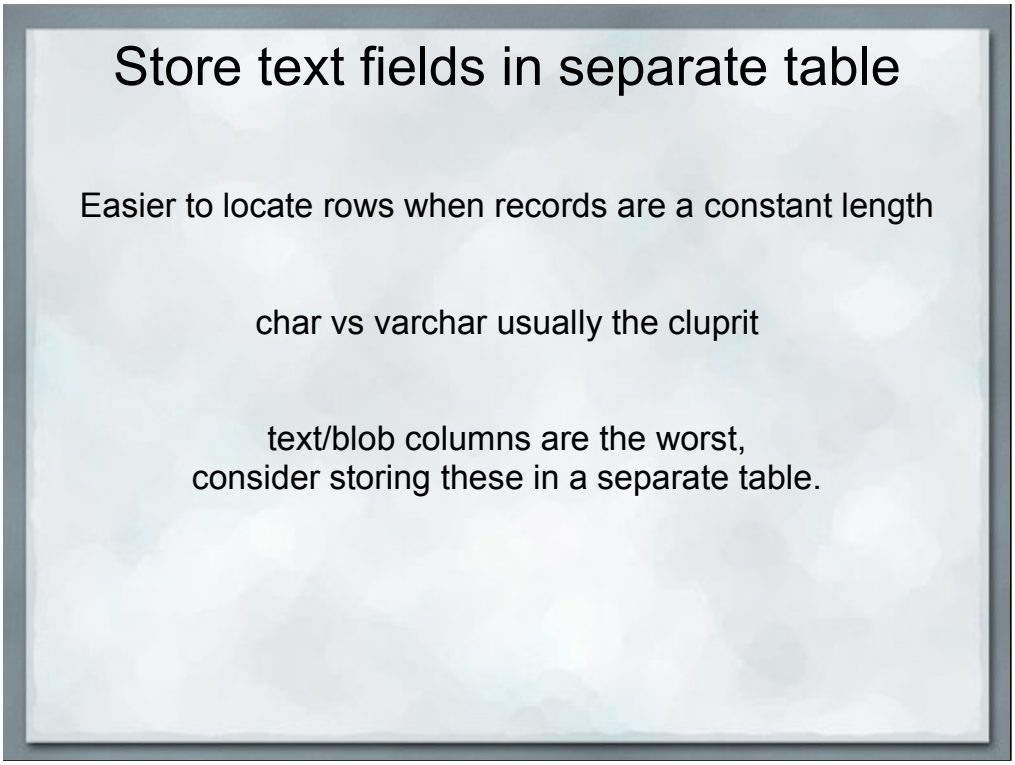
Use Multiple Queries

Sometimes it is better to break a query into multiple steps.

Instead, only get the ID values,
then fetch the bulk of the data later.

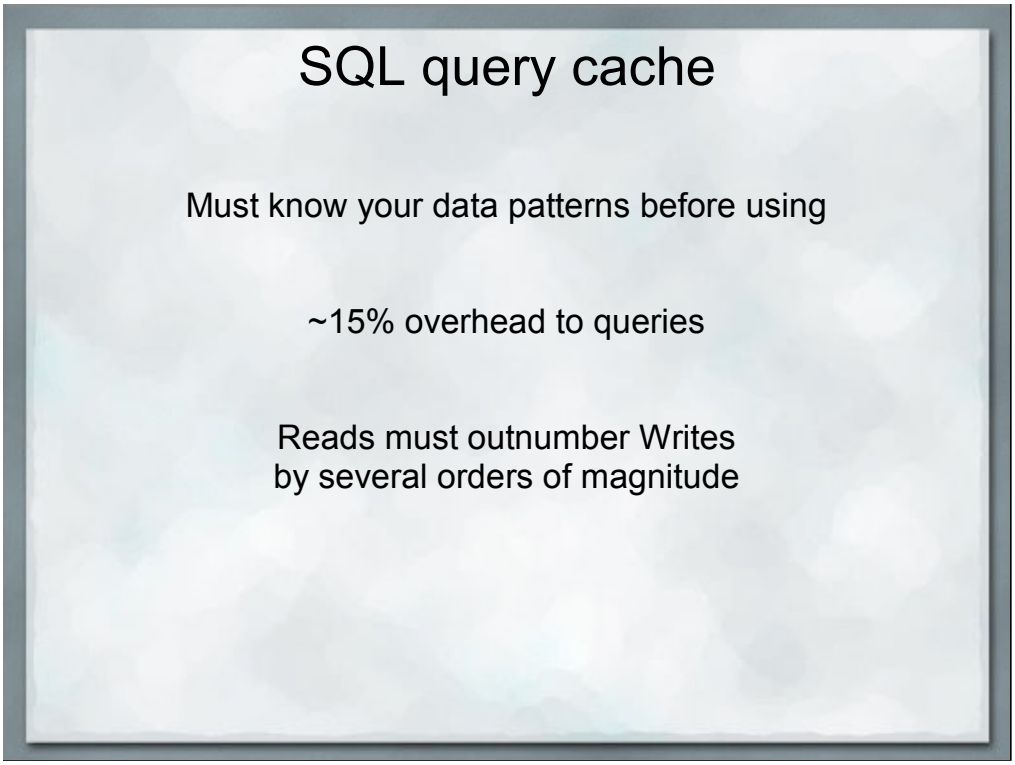
This generally only replies when you are doing both a complex query and fetching a large amount of data from each row.

BLOBs and TEXT column types can store really large chunks of data. Things like storing images inside the database.



This is an expansion on using multiple queries. Put all of the short data elements into one table, these tend to be used in the WHERE clause to filter the results.

Then put the big chunks of data in another table and fetch those later.



Any write to a table deletes all cached queries against that table.

Deadly when dealing with large joins since a write to any of the tables in the query will clear the cache.

Is SQL Query Cache Enabled

```
[wherezit]> show variables like 'query%';
+------------------------------+----------+
| Variable_name                | Value    |
+------------------------------+----------+
| query_alloc_block_size       | 8192     |
| query_cache_limit            | 1048576  |
| query_cache_min_res_unit     | 4096     |
| query_cache_size             | 67108864 |
| query_cache_type             | ON       |
| query_cache_wlock_invalidate | OFF      |
| query_prealloc_size          | 8192     |
+------------------------------+----------+
```

Must have both query_cache_type = ON or DEMAND with query_cache_size > 0 to enable the query cache.

query_cache_type = ON means that all SELECT queries are cached.

query_cache_type = DEMAND means the hint SQL_CACHE is required to cache it.

Enabling Query Cache

```
[wherezit]> SET GLOBAL query_cache_size = 64000000;
[wherezit]> SET GLOBAL query_cache_type = 'ON';

[wherezit]> SET SESSION query_cache_type = 'OFF';
```

Note that these variables will disappear when the MySQL process restarts, make changes to /etc/my.cnf to make them permenant.

Setting these variables requires the SUPER privilege as the effect is server-wide.

Note that the memory allocated is in a large chunk, preventing anything else on the system from using that memory. Possible repercussions w/ Apache or other processes.

Measuring cache performance

```
[wherezit]> show status like 'qc%';
+--------------------------+----------+
| Variable_name            | Value    |
+--------------------------+----------+
| Qcache_free_blocks       | 1        |
| Qcache_free_memory       | 59043120 |
| Qcache_hits              | 36877980 |
| Qcache_inserts           | 32656566 |
| Qcache_lowmem_prunes     | 211546   |
| Qcache_not_cached        | 6043905  |
| Qcache_queries_in_cache  | 4194     |
| Qcache_total_blocks      | 8481     |
+--------------------------+----------+
```

free_memory is large, cache is not using all of the memory allocated.

hits is not a high multipe of inserts, cache not working very efficiently.

Resetting the cache

```
[wherezit]> flush status;

[wherezit]> reset query cache;
```

flush status just resets the counts, reset query cache frees the memory and restarts the cache anew.

Tweak other server settings

MySQL Tuner - http://wiki.mysqltuner.com/MySQLTuner

Add memory to server

Use top and/or vmstat to watch disk accesses

Throw hardware at the problem

Buy/lease faster hardware
(cost of time vs cost of hardware)

Cloud computing?

High speed disk drives - Only in special circumstances

Questions?