MySQL Queries Optimization

Date and Location:	Zurich Fun tour November 21 2007
Presenting:	Peter Zaitsev, Percona

Who do we have here ?

- MySQL Developers ?
- MySQL DBAs ?
- Managers ?

Query Optimization Basics

- Avoid the query
- Cache the query
- Simplify the query
- Optimize the query

Avoid the Query

- Do you really need this query run ?
  - Web applications quite frequently run queries and do not use results
- Can you get the same results from others queries result set ?
- Can you join several queries into one ?

Cache the Query

- We assume you could not cache your page or page block which is even better
- Objects are often better to cache than query results
  - Contain results of multiple queries
  - Ready to use; save on result processing overhead
- Memcache is the leader though many variants
- Summary/Cache tables is another form of caching

Simplify the Query

- Simplify = make query to do less work for you
- Are you really using all rows query delivers ?
- What is about columns ?
- Can you possibly get rid of some joins to tables you do not use ?

Optimize the Query

- Do it when you're sure query does just what it needs to
- Adding Indexes
- Changing Query
- Using special Tricks
- There is a whole next section about it

Do we really look at it this way?

- No we often have to take things from the bottom
- Look at MySQL Traffic (log and profile query load)
  - We have great patch for MySQL 5.0 to log queries with microsecond accuracy and a lot of other info
- Find queries which
  - Are simply too long (60 sec interactive search query)
  - Queries which cause most load on the server
- Fix them
  - Change queries and schema, cache them, rework application architecture if required

Query Optimization

- Learn how MySQL Optimizer works
- Learn how MySQL Can execute queries
  - Could be it simply can't do what you want it to do
    - Especially frequent problem with Oracle converts
- Differ bad indexing from complex queries
- Have valid expectations
  - Group by of 10.000.000 rows will not be instant
- Learn to read EXPLAIN
- And profile queries to see what they do

Bad indexes or Bad Queries

- SELECT * FROM USER WHERE NAME =”Peter”
  - If this query does full table scan you have bad indexing
    - The query only needs rows with NAME=”Peter” to execute
      - Add the index
- SELECT AVG(AGE) FROM USER GROUP BY CITY
  - This is complex query which needs to traverse all rows
  - You can make it to traverse shorter index records
    - Add index on (CITY,AGE)
  - But you can't limit it to just few records
    - Will need to rethink schema and add cache/summary table

How we build summary tables ?

- Use triggers in MySQL 5.0
  - Easy to use (single place code change)
  - Reliable as no updates can slip through
  - Can get expensive for heavy updates
- Application live Updates
  - More complex and tricky but allows optimization
    - Ie keeping counts on app side and updating once
- Periodically Refresh
  - Great if you can afford a bit stale data
  - Simple and does not affect updates
  - May be too slow on large data sizes

Indexing Basics

- MySQL Can only use Prefixes of the index
- Index (A,B) can be used for
  - A=5, A=5 and B=5, A=5 and B>6
- But Can't be used for
  - B=6, B<2
- Only Equality/List allows second key part usage
  - A=5 and B>6 - will use 2 keyparts
  - A IN (1,2) and B=2 will use 2 key parts
  - A>5 and B=2 will use 1 key part only
    - B=2 will be checked while reading row/index only

Using Indexes for Order By

- Even more restricted !
- Having same index (A,B)
- A=5 ORDER BY B - Will use index
- A>5 ORDER BY B - Will not use index
- A IN (1,2) ORDER BY B – Does not help either
- A>5 ORDER BY A - Will
- ORDER BY A ASC B DESC – Does not
  - You have to do sorting in the same order for it to use index

Covering Indexes

- Very Powerful, often forgotten choice
- Allows to read data from index only, not touching data file
- Helps because index is typically smaller and it is sorted
  - (appropriate data likely needs random access)
- SELECT NAME WHERE LOGIN=”Jack234”
  - KEY(LOGIN,NAME) avoid to skip data read
- It is all or none in MySQL 5.0
  - Either all columns should be checked from the index or data will be read

LIMIT

- Limit result set. Do not fetch 100 rows if you use 10
  - Though “prefetching and caching” can be helpful ie for search applications
- LIMIT helps the most when you have index used for sorting
  - “Using Temporary”, “Using Filesort” take out most of the LIMIT benefit
- Beware of large Limit
  - LIMIT 100000,10 will fetch and discard first 100000 rows
  - Do not use LIMIT cycle in batch applications

More about LIMIT

- Limit and Rankings
- If you can precompute positions do it
  - WHERE POS BETWEEN 1001 and 1010 works much better than LIMIT 1000,10
- Beware SQL_CALC_FOUND_ROWS
  - Extra count(*) may be faster because it can often use covering index
    - Still very slow for large result sets though
- Protect your application from large limits
  - People may not go to page 500 but search engine bots well may do.

GROUP BY

- Nasty one
  - hides complexity as you get just couple of rows back
- Multiple ways to execute GROUP BY
  - Index traversal (or skip-scan)
    - If index fully matches GROUP BY Clause
  - Using temporary table
    - Good for small result set, hint SQL_SMALL_RESULT
  - Using filesort
    - Good for large result set, hint SQL_BIG_RESULT
- Can use ORDER BY NULL to avoid extra sort
  - MySQL always sorts data for group by otherwise

JOINs

- Joins are very expensive
  - 10-20 for in memory accesses, 100-1000 for disk
- MySQL only has (Optimized) nested loops joins
- If you're to traverse through a lot of rows – denormalize
  - There are other benefits as more flexible indexing strategies as well.
- “Delayed Join” - Join only to get full rows when you need

Delayed Join Example

- SELECT visitor_id, url FROM (SELECT id FROM log WHERE ip=”123.45.67.89” ORDER BY ts DESC LIMIT 50,10) l JOIN log ON (l.id=log.id) JOIN url on (url.id=log.url_id) ORDER BY TS DESC;
- Looks silly but it works well
  - Fetch from log records using covering index on (IP,TS,ID)
  - Perform LIMIT exclusion traversing index only
  - When do self join to get other columns from log table
  - And more info, such as page url from the joined url page

Sub Queries

- Can be poorly optimized in MySQL 5.0
  - Though good work in progress for MySQL 6.0
- No In-Out transformation for subqueries
  - SELECT * FROM A WHERE ID IN (SELECT ID FROM B)
- No “Caching” of non-scalar resultset even when possible
- Subselects in FROM clause result in temporary tables without any indexes

Views and Unions

- VIEWS – Never help performance
  - But can cause problems by hiding complexity
- Can be executed as MERGE or TEMORARY TABLE
- MySQL Can't push WHERE Clauses to temporary table
  - VIEW: SELECT country, COUNT(*) cnt FROM COUNTRY GROUP BY country;
  - SELECT * FROM v WHERE country=”USA”
    - Will create temporary table populate and discard the rest.
  - Same applies to ORDER BY/LIMIT w UNION
- UNION ALL also needs temporary table

That is it !

- It was short
  - But I hope you've learned something
  - Contact me with questions
    - pz@mysqlperformanceblog.com
  - Our blog
    - http://www.mysqlperformanceblog.com
  - Commercial Services
    - http://www.percona.com