■MySQLの最適化

作成：2012.11.23 yoshi

【1】DELETEのコストはかなり高い	
	読みだしがすごく多い場合は無効化を示すフィールドを作りUPDATEすべき、index更新のコストが馬鹿にならない
【2】SHOW STATUSの表示結果の解析方法	
	起動ごとに初期化、全データベースに共通
	rnd と rnd_next の割合
	Key_reads : Key_read_requests 、ディスクから読まれた回数：総回数
	この割合が1:100より悪くなったら要注意
	Key_write_requests:Key_writes 総書き込み要求回数：ディスクに書き込ま れた回数
	キャッシュの効果など
	Max_used_connections: 最大同時接続数
	Not_flushed_* : write-cacheでまだ書き込まれていない列／ブロック数
	Open_tables: Opened_tables :不必要にテーブルを開け占めするのは無駄
	== な関係が望ましい
	Slow* が増えると、デザインがなにかおかしい証拠
【3】SHOW variables like 'log*'	
	ログ内容の制御
	ログの場所 => DBと同じディレクトリもしくは my.cnf で設定
	MySQL 4.x では動作中に変更可能、mysqlコマンドから制御（数週間）
	slow_log, update_logオン, general logは普段はオフ
	slow_log は一定時間以上かかったqueryを記録
【4】slowなSELECTの理由	
	不必要なフィールド
	複雑もしくはデータが巨大
	頻繁に使われなければ最適化に過剰に手間をかける必要はない
【5】MyISM Index Structure	
	圧縮されてバランスが取れたB-Tree => O(log n)な探索パス（情報学の基本ですな）
	*.MYD => Databsae, *.MYI => Index
	インデックスの利点：小さい、メモリ内、ソート済(!)
	テキストインデックスの場合は同一プレフィックスString部分はまとめられる HogeFugeとHogeMoge, HogeMopeの場合 (Hoge) => F|Mo => g|pと検索分岐する
	多エントリで大半のフィールドが同一の値を持つ場合はインデックスの価値は小さいそういう場合には複数フィールドを要約するインデックスは劇的な効果
	複数フィールド時の順番は大事、よく考えるべき、先頭フィールドの単独検索も高速化
	index(A, B) + index(B,A) と index(A,B) + index(B) は同等だが後者の方がコンパクト、 index(A) + index(B) だと片方しか使われない
	よく使われるWHEREをよく観察すべき, 特にslow logに残るもの
	全indexの合計がDBデータより大きくなっても、それぞれのindexツリーはずっと小さい
	indexの数が多すぎるとUPDATEおよび特にINSERT, DELETEが遅くなる、SELECTごと にどのindexを使うかの決断が必要
	文字列のソート順はサーバ単位で統一、MySQL4だとフィールドごとに設定可能
【6】MySQLがindexを活用する時	
	フィールド値を定数と比較するとき（WHERE name = ″hogehoge″ ）
	フィールド値全体でJOINするとき（ WHERE a.name = b.name ）
	フィールド値の範囲を求める時 >,
	LIKEで文字列の先頭が固定な時
	MIN(), MAX()（複数要素indexの同一first fieldでsecond fieldのmin,max でも有効）
	文字列のプレフィックスをもとにしたORDER BY, GROUP BY
	WHEREのすべてのフィールドがindexの一部の場合（DBまったく参照されず）
【7】indexが使われない時	
	LIKEがワイルドカードで始まる時
	DB全体を読んだ方が早いとMySQLが判断した時
	通常はindexはORDER BYには使われない
	WHERE と ORDER BYのフィールドが違う時にはどちらかしか使われない

【8】ボトルネックとなるSELECT => EXPLAIN SELECT... ¥Gで解剖	
	どういう順番でテーブルとどのインデックスが読まれるか表示
	rowでそのテーブル読まれた行の数（少ない方が良い）
	rowが多いテーブルを最適化すべし（インデックス作成）
	typeの値（早 遅）: system, const(結果が単一行), eq_ref(UNIQUE or PRIMARY index使用), ref（index使用の単一パス検索）, range（ indexの範囲検索）、 index(index全体をスキャン), ALL（全DBデータを検索）
	Extrasで見たくないもの： using filesort（余分なソート）、using temporary（一時ファイルの作成）
	Extrasで見たいもの：using index（DB本体を読む必要なし）、Where used（type: ALLとの組み合わせだとindex作成推奨）
【9】SELECTの最適化	
	LEFT JOIN, STRAIGHT JOINとUSINGの組み合わせはWEREより早い
	WHERE field INはかなり早い
	WHEREが使えるならHAVINGは使うな（HAVINGはindex使わず）
	なるべく簡単なステートメントを（オプティマイザが働きやすい）
【10】UPDATE, INSERT, DELETEの最適化	
	SELECTと同じルール
	multi-row INSERTをなるべく活用
	DELETEとUPDATEをなるべく避ける、(>=1000query / secの場合）
	INSERT DELAYEDは徹底的に避けるべし、=> LOCK TABLE xx READ LOCAL
【11】MySQL独自の最適化拡張	
	SELECT HIGH_PRIORITY 優先度をあげる
	UPDATE LOW_PRIORITY
	REPLACEのほうがDELETE -> INSERTより良い、indexの再利用、先にDELETEするかの判断をMySQLに委任
	LOAD DATA INFILE / LOAD_FILE()
	LIMIT # => データ転送量の削減
【12】DBスキーマの最適化（MySQL特有）	
	JOINに使われるフィールドは完全同一形式を
	SELECT ... PROCEDIRE ANALYSE() ¥G => 最大必要フィールド長を分析
	不必要に長いCHAR, VARCHARを避けるべし 資源の無駄使い
	ENUMを使う => エントリごとに1バイトのみ必要 =>高速化、内容の正しさチェック
	NULLがいらないフィールドはNOT NULLを使用
	なるべくVARCHARでなくCHAR、VARCHARは別テーブルに（全フィールドの固定長化）
	SHOW table status like 'hoge'; で row_format == Fixed
	UPDATEの頻度やトランザクションの有無でテーブルタイプを選択 HEAD, MyISAM, InnoDB
	読みだしオンリーなテーブルは圧縮（myisampack(1)）
【13】mysqldのメモリ使用	
	key_buffer_size: indexの常駐量（総RAMの1/4推奨、デフォルトはたった8）
	table_cache：なるべく多く、増やしすぎるとスワップされるので注意
	スレッド固有のメモリ値を大きくしすぎるとmalloc()に時間がかかりすぎ、スレッド作成が遅くなる
【14】OSの正しいチューン	
	正しいファイルシステム => ReiserFS推奨
	DBが多い場合は max open filesをアップ
	最大スレッド/fork()数をアップ（でないとスケジューラペナルティ）
	なにがなんでもスワップを避ける
	NFSを避ける
	多数のハードディスクに分散
	シークタイムの短いハードディスクを使用、転送スピードより大事
	ハードウェアの優先順：とにかくRAM, HDD, CPU, Network

参照先：http://slashdot.jp/~Oliver/journal/26710