1

MySQL の特徴

takeshi@mysql.gr.jp

2004-08-20

2

History

- 1979 Monty が UNIREG 書く (メモリ 32kB のマシンで動作)
- 1994 MySQL 開発開始。(Innobase Oy 設立)
- 1995 MySQL 1.0 誕生。既にマルチスレッド構造
- 1996 MySQL 3.11 インターネット上に公開。LIMIT 文
- 1996 Perl, PHP/FI インターフェース登場
- 1997 日本語対応 (ujis,sjis 。とみた氏) 。REPLACE 文
- 1998 MyNA ML スタート (このときはまだ MyNA ではない)
- 1998 Ruby インターフェース (とみた氏)
- 1999 HEAP, MyISAM テーブル
- 2000 MySQL AB 設立。(それまでは TcX Data Konsult AB)
- 2000 レプリケーション、RAID 、MERGE 、tcp-wrapper 、EMIC クラスタ
- 2001 InnoDB(トランザクション) 、SSL 通信、クエリキャッシュ
- 2003 副問い合わせ、ストアドプロシジャ、OpenGIS
- 2004 NDB クラスタ

3

全体像

- マルチスレッド
  - **CPU** を増やすだけで性能アップ。開発当初から採用した構造。
- データの保存形式 **(** ストレージエンジン **)** が **10** 以上。
  - 拡張性、保守性のアップ。最適化の簡易

- 設定無しで動作
- 簡単インストール。バイナリをコピーするだけ。
- アップグレード時にフルダンプ、フルリストアが不要。前のデータはそのまま使用できる
- データはファイルに記録される

- 多くの **OS** で動作。**MS-Windows** はネイティブ

4

全体像

- レプリケーション **(>= ver.3.23** 一方向。両方向は開発中 **)**
- **NDB** クラスター **(>= ver.4.1)**
- **EMIC** クラスター **(>= ver.3.23)**
- **SSL** 通信 **(>= ver.4.0)**
- **tcp-wrapper (>= ver.4.0)**

- クエリキャッシュ **(>= ver.4.0)**
- **FULL TEXT INDEX (>= ver.4.0** ただしシングルバイト文字 **)**
- 副問い合わせ **(>= ver. 4.1)**
- ストアドプロシジャ **(>= ver. 5.0)**
- **OpenGIS (>= ver.4.1)**

5

全体像

- トランザクション **(InnoDB) (>= ver.3.23)**
  - マルチバージョニング
  - **4** つのトランザクション分離レベル
  - **SAVEPOINT, ROLLBACK TO** ***savepointname***
  - 行レベルロック
- 更新クエリだけを記録するログ
- 処理に時間のかかったクエリだけを記録するログ
- 多くのマルチバイト・キャラクタ・セット **(ujis,sjis,utf8,...)**
  - **ucs2, utf8 (>= ver.4.1)**
  - データベース、テーブル、フィールド単位にキャラクタ・セットを指定可能 **(>= ver.4.1)**
  - サーバー、クライアント間での文字コードの自動変換 **(>= ver.4.1)**

6

全体像

- 多くの言語インターフェース
  - **C/C++, Ruby, PHP, Perl, Java, ODBC, Python, tcl...**
  - 全ての言語でエラーコードの取得は可能
- 有名な **MySQL** 対応アプリ
  - **JBoss, XOOPS, PHP-nuke, phpmyadmin, Doblog, StarOffice, Apache** 認証やログ **, postfix, qmail, freeradius, proftpd, nagios** など
  - **ER/Studio6.x**
- サーバー部分だけがライブラリになっている。サーバーの埋め込みが可能。**(>= ver.4.1)**

7

MySQL サーバー

- **mysqld** がサーバーのバイナリ
- **mysqld 1** つのバイナリが接続受付、クエリの処理、ファイルの操作を行う
- **mysqld** はスレッドに処理を任せる
  - **1** クライアントにつき **1** スレッドが処理を担当
- **100** のクライアントからの同時接続要求を **50** ～ **70msec.** 程度で完了
- **CPU** を足すだけで性能アップ。その際、何らかの変更は不要。

- サーバーの心臓部はライブラリになっていて、組み込み利用が可能
  - ライブラリの組み込みは簡単。サーバーライブラリを使うためには **2** つの関数を一度使用するだけ。あとは **MySQL** クライアントライブラリ関数と変わらない。

8

マルチ・ストレージエンジン

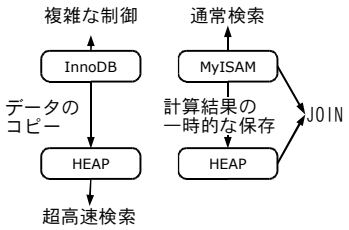
- ストレージエンジン **=>** 保存形式、テーブル型
- ストレージエンジンで機能、能力が変わる
- ストレージエンジンの特性を活かすことで、最適な環境が実現
- 今後新しい考え、手法が登場したとき、
  - 簡単に新しいストレージエンジン、新しい機能を追加できる
  - 前のストレージエンジンのファイルは引き続き使用できる
  - バグの切り分けのしやすさ。互いに影響しあわない

9

各ストレージエンジンの特徴

- 比較図

	MyISAM	InnoDB	HEAP	NDB
保存メディア	disk	disk	memory	memory
ロック	テーブル	行レベル	テーブル	行レベル
トランザクション	なし	あり	なし	あり
記録量上限	2^64バイト/1テーブル	64Tバイト/全体	メモリ上限	メモリ上限
特徴	速い	トランザクション	速い	クラスター
主な用途	検索が多い場面	トランザクション	一時的な記録 高速な検索	クラスター

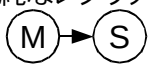


10

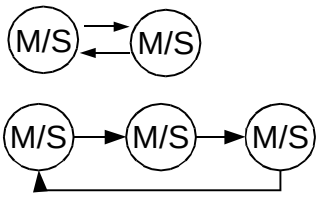
レプリケーション

- 代表的な構成

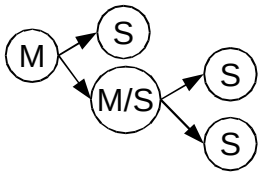
単純なレプリケーション

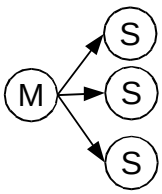


ちょっと変わった構成



SELECT の負荷分散
検索の多い Web サイトなど

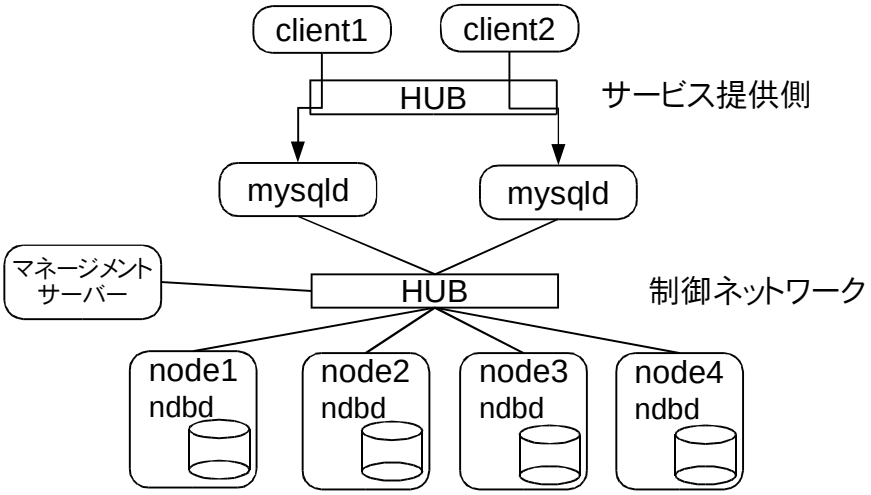




11

NDB クラスター

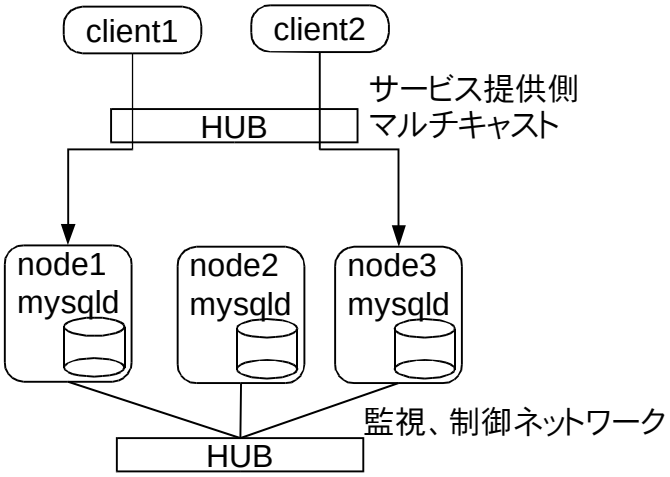
- メモリ内にデータを保存



12

Emic アプリケーションクラスター

- 概念図



13

通信

- **SSL** 通信
  - **OpenSSL** ライブラリを組み込むことにより、通信経路を暗号化できる
- **tcp-wrapper**
  - **tcp-wrapper** ライブラリを組み込むことで、**MySQL** の認証機構を使用する前に、制限ができる。
- **MySQL** ユーザーのパスワード
  - 不可逆なパスワード

14

SQL 文

- クエリキャッシュ
  - 素早い反応が期待できる
- 副問い合わせ
- ストアド・プロシジャ
- **OpenGIS** ライブラリを使用した、**GIS** のサポート
- **MySQL** の便利 **SQL** 文
  - **LIMIT**
  - **FULL TEXT INDEX** を使用する文 **(MATCH, AGAINST)**
  - **REPLACE INTO**
  - **CREATE TABLE .... SELECT ....**
  - **ALTER TABLE**
    - データがテーブルにある状態で、フィールドの追加、変更、削除、テーブル名の変更、フィールド名の変更、ストレージエンジンの変更ができる。
  - **EXPLAIN, USE INDEX, IGNORE INDEX**
  - 日付や文字列操作に関する関数が多い
  - **LOAD DATA INFILE .... IGNORE** 行数 **LINES**

15

トランザクション (InnoDB)

- マルチバージョニング
  - ロック待ち無しでの読み取りが可能
- 標準の **isolation level** は **REPEATABLE READ**
  - **InnoDB** の **REPEATABLE READ** では、ファントム・リードは発生しない
- 行レベルロック
- ファイルは分散配置が可能
  - 必要とされるファイルは、ばらばらのディレクトリに置くことが可能
  - テーブル単位にファイルを分けることも可能 **(ver.4.1.1** 以上 **)**
- **Innobase Oy** が開発

16

SQL 文を記録するログ

- 更新クエリーだけを記録するログ。バイナリ更新ログ。
  - ログを採取しはじめる時点のフルバックアップとバイナリ更新ログがあれば、ある時点のデータの状態にまで戻ることが可能。
  - **--log-bin**
- 処理に時間のかかったクエリだけを記録するログ。スロー・クエリー・ログ
  - どのクエリがネックかわかる
  - **--log-slow-queries**
- インデックスを使用しなかったクエリーだけを記録するログ。
  - どのクエリがネックかわかる
  - **--log-queries-not-using-indexes**

17

マルチバイトのキャラクター・セット

- **ujis,sjis** は **ver.3.21** から利用可能
- **ver.4.0** まで
  - 使用できるキャラクター・セットは、**mysqld 1** つにつき **1** つのみ
  - **char(10)** は **10** バイト
- **utf8, ucs2** は **ver.4.1** から
- **ver.4.1** から
  - データベース単位、テーブル単位、フィールド単位にキャラクター・セットを指定できる。
  - クライアントとサーバーのキャラクター・セットが違う場合は、文字コードの変換が自動で行われる。
  - **mysqldump** が出力するコードの標準は **utf8 (** オプションで変更可 **)**
  - **char(10)** は **10** 文字
  - **binary(10)** は **10** バイト

18

3.X,4.0 => 4.1

- 今から **MySQL** を使用しはじめる人は **ver.4.1** を最初から使う。
- **MySQL ver.4.0** までを現在使用している人は、**MySQL ver.4.1** 以上にするときは注意が必要。
- **char()** の仕様の違い
  - **1** バイトしか使っていない人は関係ない。困るのはマルチバイトを使用している人。
  - **ver.4.0** までは **char(10)** は **10** バイト。**ver.4.1** 以上では **char (10)** は **10** 文字。
  - **ver.4.1** に上げてから **ALTER TABLE** する
    - **MyNA** でそのための **shell** スクリプトを提供
  - **ver.4.1** に上げる前にダンプしておき、**ver.4.1** にしてからリストアする。

19

3.X,4.0 => 4.1

- **MySQL ver.4.0** までを現在使用している人は、**MySQL ver.4.1** 以上にするときは注意が必要。

- パスワードの仕様の違い
  - **ver.4.1** からはパスワードの保存形式が変わった。
  - **ver.4.0** までのパスワードが記録されていても、そのまま問題なく運用可能。なにか特別なことをする必要は無い。
  - **ver.4.1** に上げて、**ver.4.0** までのパスワード形式で運用したい場合
    - **old-passwords** オプションをサーバーに与える。
  - **ver.4.1** に上げて、新しいパスワード形式で運用したい場合
    - クライアントプログラムを、**ver.4.1** の **libmysqlclient** でリンクし直す

20

3.X,4.0 => 4.1

- **MySQL ver.4.0** までを現在使用している人は、**MySQL ver.4.1** 以上にするときは注意が必要。

- **mysqldump** の仕様の違い
  - **ver.4.1** からは、**mysqldump** は出力する **SQL** 文を **utf8** にしてしまう
    - **mysqldump --default-character-set=ujis** でダンプするようにする

21

4.1 の文字コード自動変換

- クライアントとサーバーのキャラクター・セットを合わせて運用するのが **best**

- もしクライアントとサーバーのキャラクター・セットが違う場合、文字の破壊が起こる可能性がある。**(** マルチバイト・キャラクター・セットとシングルバイト・キャラクター・セットが混在したとき **)**
- 全ての言語の **MySQL** モジュール **(ex.PHP,Perl,Ruby,C,...)**、全ての **MySQL** クライアントは、リンクしている **MySQL** クライアントライブラリの標準キャラクター・セットに注意する必要がある。
  - 海外でコンパイルされた **libmysqlclient** や **libmysql.dll** は、どんなキャラクター・セットが採用されているかわからない。**(** ほとんどは **latin1** だろう **)**
    - それを知らないでそのバイナリを使用すると文字破壊がおきる可能性大

== メンテナーの方には注意して欲しい **!==**

22

4.1 の文字コード自動変換

- 文字破壊を避けるには **...**
  - 例えば **PHP** スクリプトを書き直して、キャラクター・セットを告げる **SQL** 文を書き足すことで回避できるが **....**

    → **今動いているアプリの全てを書き直すなんて、やってられない！**
  - 例えば **my.cnf** に **default-character-set** を書き足すことで回避できるが **...**
    - 残念！ **PHP4.3.8** は **my.cnf** を読まない。**Perl** でも **Ruby** でも **PHP5.0** の **mysqli** でも、**my.cnf** を読むためにスクリプトの変更が必要

      → やってられない

⇒ クライアントライブラリのキャラクター・セットを **binary** にする

- サーバーのキャラクター・セットが何であれ、文字が破壊されることは無い **!**
- 設定の変更もスクリプトの変更も不要 **!**
- **MyNA** で **4.1.3-beta** 用のパッチを提供。

  ./configure --with-client-charset=binary --with-charset=ujis
  - これでできた **MySQL** クライアントライブラリをリンクする
- ただし、データベース名、テーブル名、フィールド名にマルチバイト文字は使用できない

23

4.1 の文字コード自動変換

- **4.1** 文字コード自動変換機能の問題に対する **MySQL AB** のスタンス。

以下については対応する気は無いらしい

  - クライアントライブラリを **binary** でコンパイルすること
  - クライアントだけを違うキャラクター・セットでコンパイルできるようにすること
  - **binary** を使用したときに **DB** 名、**table** 名、**field** 名にマルチバイト文字が使えないこと
  - **mysqldump** が標準で **utf8** で出力してしまう **(** コンパイル時のキャラクター・セットの指定を無視して **)** こと。

- 対応希望項目に対しては、要望をどしどし上げていきましょう **!**
  - **http://www.mysql.gr.jp** の「バグ報告」
  - **MyNA ML**
  - **http://bugs.mysql.com**