MySQL 概要

まいえすきゅーえる

日本 MySQL ユーザー会
MySQL Nippon Association
http://www.mysql.gr.jp/

第 2 版 : 2005-07-29 JPUG
第 1 版 : 2004-10-22 KOF

MyNA History

黎明期

- **1997** とみた **(** 当時のハンドル名は民斗 **)** 氏による日本語対応パッチ **(ujis,sjis)** および日本語ドキュメント
- **1998 ML** スタート

本格稼働

- **2000 MyNA(MySQL Nippon Association)** に名称決定。**mysql.gr.jp** 取得。サーバー設置
- **2003** 初めてのユーザー会の勉強会
- **200401** 初めての **developer summit**
- **200408 OSC** 参加
- **200409 LinuxWorld** 参加
- **20041120** 第2回勉強会
- **20050325,26 OSC** 参加

3

MyNA ではなにをやっているか？

- ML, Web
- 勉強会
- イベント参加
  - 今, LinuxWorld, OSC
- bug, 要望取りまとめ
- 執筆依頼があったら募集
- 飲み会

体制

- 会長 (とみた氏) 、副会長 (たてやん氏、堤井氏) 、Webmaster(坂井氏)
- サーバーは借りてます
- 金は集めてません
- 事務局等、組織としての形は全くありません。

4

MySQL 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 ,Windows(もっと前かも)
- 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(MEMORY), 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 クラスター
- 2001 InnoDB(トランザクション) 、SSL 通信、クエリー・キャッシュ
- 2003 副問い合わせ、ストアドプロシジャ、OpenGIS
- 2004 NDB クラスター
- 2005 view, trigger, federated

5

MySQL 特徴 1

- 高速な動作
  - **MySQL** は素早いレスポンスを最も重要な目標として掲げています。マルチスレッドで処理を行います。
  - 主要 **DB** サーバにて行なったベンチマークテスト **(eweek** の評価 **)** では、**MySQL (InnoDB)** は **Oracle** に匹敵する評価を得ています。
- 手軽に使える
  - インストール、運用はとても簡単に出来ます。
  - データはファイルに格納されているため、バックアップや移動を簡単に行なうことができます。
- 世界中に普及
  - 世界で最も人気のあるオープンソースデータベースです。
  - 米「 **Linux Journal** 」誌の読者投票では **6** 年連続で最高のデータベースに選ばれています。
- 多くのプラットフォームをサポート
  - **Linux, Microsoft Windows, FreeBSD, Sun Solaris, AIX, Mac OS X, HP-UX, QNX, Novell NetWare, SCO OpenUnix, SGI Irix, Dec OSF** といった様々なプラットフォームに対応しています。
  - 特に **Windows** 版は、初心者にとって心強い存在になっています。

6

MySQL 特徴 2

- オープンソース
  - **MySQL** は、「オープンソースの商用データベース」です。公開当初からソースが公開され、社員だけではなく、全世界のボランティアの手で機能の拡張や修正が行なわれています。
- 多言語対応
  - データはもちろん、テーブル名やフィールド名にも日本語を使用できます。また、文字列処理関数の多くがマルチバイト対応され、便利に日本語を利用できます。
- 選択可能なライセンス形態
  - **GPL** とコマーシャルライセンスが選択できます。**GPL** が条件にあわない場合は、コマーシャルライセンスを選択して **GPL** を回避することができます。**(** 商用利用するかどうかは関係ありません。あくまでも **GPL** を承諾するか否か **)**
- 多種の対応言語
  - **C** 、**C++** 、**PHP** 、**Ruby** 、**Perl** 、**Python** 、**Java** 、**ODBC** 、**Tcl** などの多くの言語で **MySQL** を利用できます。
  - 特に **PHP** との組み合わせでよく使われており、**Linux + Apache + MySQL + PHP** の環境を表す **LAMP** という言葉までできているほどです。
  - **ANSI SQL 92** に準拠しています。

7

MySQL と MySQL AB

- **MySQL** を開発、メンテナンスしているのは、**MySQL AB** というスゥエーデンの会社です。開発者は世界中に何十人といます。
- ソースの保証
  - **MySQL AB** は、ソースの保証をしています。
  - ユーザーからの修正、提供パッチは必ず **AB** の人間がライセンスをチェックしています。
    - ライセンスや特許侵害における心配はありません。
- **MySQL** の向かうところ
  - バグが無いこと
  - 楽である、簡単であること
  - 使って楽しい
- **MySQL AB** 自体の売上は、毎年 **2,3** 倍で増えているらしい。
  - それだけ世界に認められているらしい。開発者も多いらしい
  - **NASA, Yahoo, google,** はてな **, mixi,** 多くのプロバイダー **, ...**
  - **ER/Studio ver.6** が **MySQL** に対応。

8

MySQL のライセンス

- **GPL** かコマーシャルライセンスか
  - **GPL** を選択するなら費用は **0** 円
  - **GPL** を選択しないならコマーシャルライセンスを購入すれば **OK**
  - **PHP** と合せて使用する場合は、特例のライセンスがあります。**(0** 円 **)**
  - 商用利用するかどうかは関係ありません。
  - 学校で使う場合は、**MySQL AB** に言えばコマーシャルライセンスでも **0** 円になる場合があるらしい。
  - **2000** 年にこのライセンスに変わりました。それ以前のライセンス条項とは別物です。古い情報で判断すると損ですよ。
- **OS** によるライセンスの違いはありません。
- **CPU** 数、接続クライアント数は無関係です。

9

MySQL の製品ラインナップ

- **MySQL**
  - 通常、**MySQL** と言えばこれ。
  - 高速検索と、トランザクションをサポートしています。
  - **NDB** クラスターは **MySQL 4.1** 本体に含まれています。**MySQL AB** が **Alzato** から引き継ぎました。
  - **MySQL-Max** は、**MySQL** の使用できる機能の全てを組み込んだバイナリを指します。**MySQL** と違う物ではありません。
  - **MS-Windows** 用 **(** ネイティブ **)** もずいぶん前からあります。
- **MaxDB**
  - **SAP DB** です。**MySQL AB** が **SAP** から引き継ぎました。
  - 現在、**MySQL** そのものとは統合されていません。将来的には一つにする予定らしいです。

10

MySQL の製品ラインナップ 2

- **MyODBC (Connector/ODBC)**
  - **ODBC** インターフェースです
  - **MS-Windows** 用と **Unix** 用があります。
- **Connector/J**
  - **JDBC** インターフェースです。
- **mysql-administrator**
  - **GUI** の管理ツールです。
  - **MS-Windows** 用と **Unix** 用があります。

11

MySQL の今と予定

- **4.0**
  - クエリ・キャッシュ、**SSL, tcp-wrapper, ujis, sjis**
- **4.1 (** 現在リリース **)** ← **4.1** 使うくらいなら **5.0** を使う
  - **utf8, ucs2, sjis, cp932, ujis** サポート
  - 文字コードの自動変換
  - データベース、テーブル、フィールド単位に文字コードを変更できる
  - 副問い合わせ、**OpenGIS** サポート
  - **NDB** クラスター
- **5.0(** 現在開発中 **beta** 状態 **)**← 今から使うなら **5.0** を使う
  - ストアド・プロシジャ
  - トリガー
  - ビュー
  - **FEDERATED** エンジン： リモートの **MySQL** が管理しているテーブルをあたかもローカルにあるように見せる

12

MySQL の活動フィールド

- 適材適所
  - 機能があるというだけで製品を選んでいませんか?
  - 全ての機能が本当に必要ですか?
  - 同時 **100** セッションの確立は、**60msec.** 程度 **(100baseT)** で完了します。
    - これだけ速ければコネクションプーリング自体いらないよね。今までの観念が壊れます。
  - 必要なものだけを、少ない投資で得られればそれでいいのでは?
    - うん千 **(** 百 **)** 万円 対 数万円 **(0** 円 **)**
  - すべての **DB** を **MySQL** に置き換えれるとは思ってません。が、効果的に使えればいいのではないですか? 所詮道具です。
- **MySQL** はソースもバイナリも **Web** で公開されています。
  - ぜひダウンロードして試してください。
- **MySQL AB** という会社の保証付き

ひと休み

13

今回のあるある

WHERE をわすれて 全て消す

とある SNS アバター集より

14

MySQL の手軽さ1

- 設定ファイルなしでも動きます。
- **32M** もあれば動きます
- バイナリインストール
  - **Unix** も **Windows** もバイナリパッケージを展開するだけ。
    - バイナリをコピーすれば動くということです。
  - **MS-Windows** 用 **MySQL** はレジストリを操作しません。
  - **MS-Windows** 用 **MySQL** は、**NTFS, VFAT** 共に動作します。
  - **Unix** のバイナリは、どのディレクトリー、どのアカウントでも動きます。
    - **3** 個から **5** 個のオプションの指定で自由に
- **CPU** を増やすだけで性能 **UP**
  - **CPU** を追加したとき特別な変更作業は発生しません。
  - もともとマルチスレッドなので無理がない
- 最初っから内部は **64bit** 対応

15

MySQL の手軽さ2

- バージョンアップはバイナリを差し替えるだけ
  - 古いファイルはそのまま使用できます。フルダンプの必要はありません
    - **3.23** のデータの状態で **5.0** にバイナリを差し替えても動く。
    - 逆も動く。
- データファイルはコピーして違う機械に持っていくだけで **OK**
  - 機種依存性はほとんど無い
- フルダンプや差分の記録は **SQL** 文で記録される
  - その記録を違う環境に持って行って、その **SQL** を実行すれば、たやすく同じ状況が再現できる

16

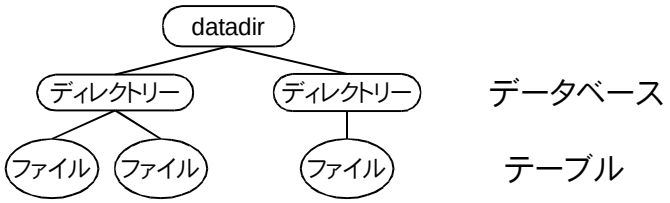
MySQL の手軽さ3

- 多くのバイナリ配布のソフトが最初から **MySQL** 対応
  - ほとんどの **Linux** ディストリビューションには **MySQL** のバイナリパッケージがあります。
  - **MacOSX** にも付属しています。
  - **Perl,PHP,Ruby** なども、だれかがバイナリを作っています
- 多くのレンタルサーバーにも **MySQL** がインストール済み
  - 海外は多いです
- 情報量
  - 日本語書籍 **25** 冊以上 **?**
    - 翔泳社「**MySQL** 徹底 ***** 」、ソフトバンクパブリッシング「実践 **MySQL4** 」
  - 日本語マニュアル
  - 雑誌記事や **web** の記事が増えてきた
    - **Web+DB vol22** や **DB magazine** 連載は濃い内容
    - 昔に比べればかなり使われてきてきたのかも
    - 間違った記事や投稿もある **(** 普及のしるしか? **)**

17

MySQL の手軽さ4

- シンプル
  - 必要なものだけがあればいい
  - ディスクイメージじゃなくて **SQL** 文でログが出る
  - データベース **=directory,** テーブル **=file**
    - データベースを示すディレクトリーに別のパーティションを **mount** してもよし、
    - データベースを示すディレクトリーをシンボリックリンクにしても良し



18

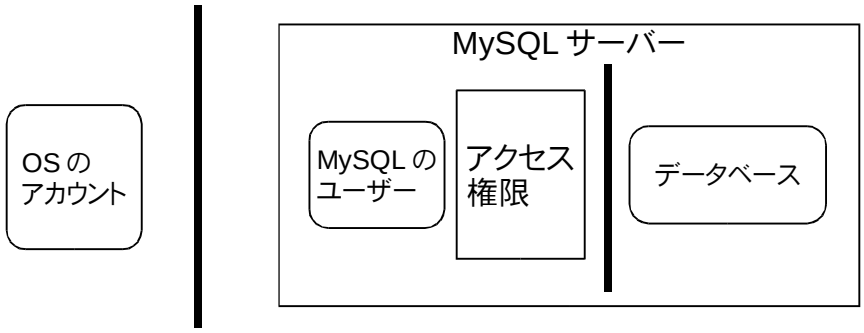
MySQL の手軽さ5

- **Unix/Linux** に慣れた者にはうれしいインターフェース。コマンドの組み合わせで動作させることが多い
  - **mysqlbinlog filename | mysql -h** ホスト名 **-p**
  - **tail -f** 詳細ログファイル **| logger**
  - **export MYSQL_PWD="** パスワード **"**
    echo "SELECT" | mysql -u ユーザー名
  - **mysql -uusername -p** パスワード **-e 'SELECT'**

19

データベースとユーザー

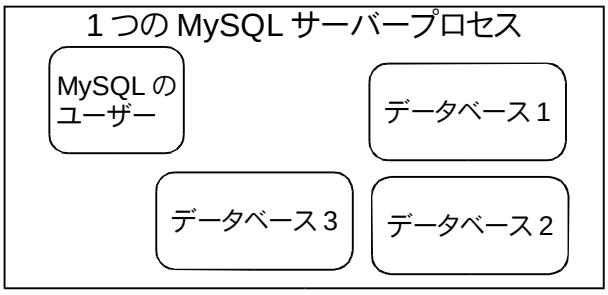
- 他の **DB** に比べて、データベースとユーザーの結びつきは強くありません。
- データーベースの所有者という概念はありません。
- データベースに対するアクセス権限がユーザーに与えられているだけです
-
- **OS** のアカウントと、ユーザーやデータベースも関係ありません



20

データベースとプロセス

- データベースとプロセスは関係ありません。
- **1** つのプロセスが、複数のデータベースを管理しています。
-
- **1** プロセスが全てやります
  - **Listen**、解析、最適化、**I/O.....**
  - 各処理はマルチスレッドで分担



21

クエリー・キャッシュ

- **SQL** 文を覚えます
  - スペース、大文字小文字も入れて、完全に文が一致しないとヒットしません
- **SQL** 文を実行した結果を覚えます

22

組み込み MySQL サーバー

- サーバーの心臓部がライブラリーになっています。
- **C/C++** プログラムに組み込みが可能です。
- 使い方は簡単。普通のクライアントプログラムに関数 **2** つを加えるだけ。あとは特別なことはしません。
- **SQL** 文も解釈します。
- 多くのストレージエンジンも持っています。
- ネットワーク経由の接続機能は持っていません。

23

MySQL のメモリー使用

- 共有メモリー
  - クエリーキャッシュ
  - MEMORY(HEAP) テーブル
  - MyISAM インデックスのキャッシュ
  - InnoDB のバッファ。レコード、インデックス、ログ、システム情報
- スレッド単位に割り当てられるメモリー
  - sort 時、join 時に使うバッファ
  - MyISAM レコードのキャッシュ
  - TEMPORARY テーブル

24

マルチバージョニング

- **InnoDB** ストレージエンジンで実装

25

マルチランゲージ

- 複数のキャラクターセットが 1 つのバイナリ、1 つのプロセスで複数使用できる。
  - binary armscii8 ascii big5 cp1250 cp1251 cp1256 cp1257 cp850 cp852 cp866 cp932 dec8 eucjpms euckr gb2312 gbk geostd8, greek hebrew hp8 keybcs2 koi8r koi8u latin1 latin2 latin5 latin7 macce macroman sjis swe7 tis620 ucs2 ujis utf8
- データベース、テーブル、フィールドそれぞれにキャラクターセットを指定できる
- サーバーとクライアントのキャラクターセットが違う場合、自動的に文字を変換する (サーバーが変換する)
- OS の関数を使用していないので、動作は OS 非依存で皆同じ。

26

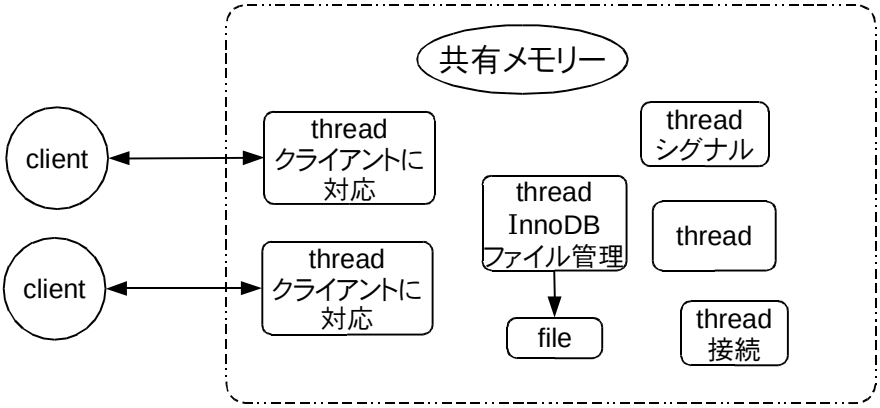
マルチランゲージ

- 多くの開発言語のインターフェースがある
- C, Ruby, PHP, Perl, Python, Java, Tcl.....
- C の API を元にした関数 (メソッド) 名なので、一つやれば他の言語はすんなり書ける
-
- C では、SELECT クエリーの結果は、どのフィールド型でも、値は *char で返ってくるので、汎用的なプログラムを書きやすい
  - 最近できた Prepare 関数は、フィールドの型と変数の型を合せないといけないので書きにくい
  -
- エラーコードとエラーメッセージは全ての開発言語で取得できる

27

マルチスレッド

- 各クライアントにつき、**1** スレッド立ち上げて対応
- 内部の **I/O** 処理などは別のスレッドが対応



28

マルチストレージエンジン (テーブル型)

- **MySQL** 独特の特徴
- 最適化
  - 使い分けを行うことで、特別なチューニングを施すことなく、簡単によりよい状態にすることが可能。
- 拡張性
  - 今後新しい考え、手法が登場したとき、簡単に新しい機能を追加できる。しかも今までの部分に影響はでない。
- 互換性
  - 前のストレージエンジンのファイルは引き続き使用可能。
- 保守性、信頼性
  - バグの切り分けのしやすさ。他のエンジンに影響を及ぼしません。

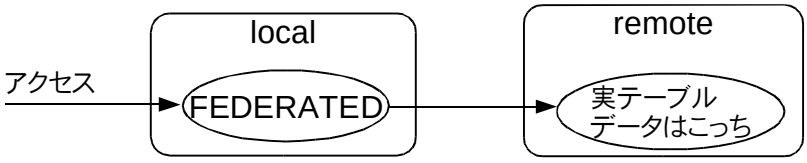
接続受付 , クエリの解析 , 最適化等					上位層
MyISAM	**InnoDB**	**MEMORY**	**NDB**		下位層 ストレージ エンジン

各ストレージエンジン概要

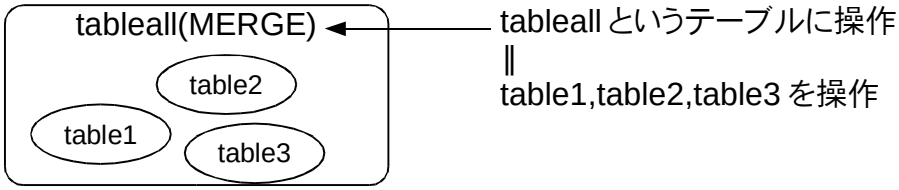
	MyISAM	InnoDB	MEMORY	NDB
保存メディア	disk	disk	memory	memory
ロック	テーブル	行レベル	テーブル	行レベル
トランザクション	なし	あり	なし	あり
分離レベル	なし	4つ	なし	1つ(RC)
記録量上限	2^64バイト/1テーブル	64Tバイト/全体	メモリ上限	メモリ上限
特徴	速い	トランザクション	速い	クラスター
主な用途	検索が多い場面	トランザクション	一時的な記録 高速な検索	クラスター
マルチバージョニング	なし	あり	なし	なし

他のストレージエンジン

- FEDERATED ストレージエンジン (version 5.0 から)
  - リモートの MySQL サーバーにあるテーブルを、あたかも LOCAL のテーブルのように見せかける



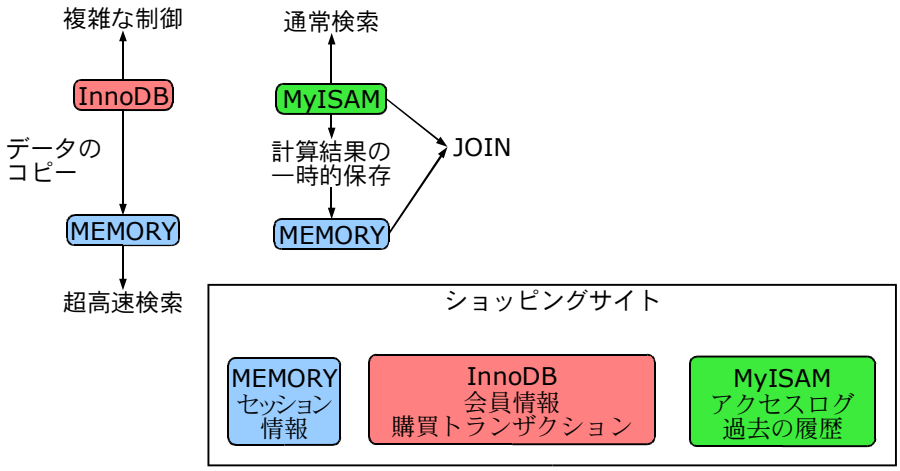
- MERGE
  - 複数のテーブルを１つのテーブルに見せかける



31

各ストレージエンジン (テーブル型) の使い分け例

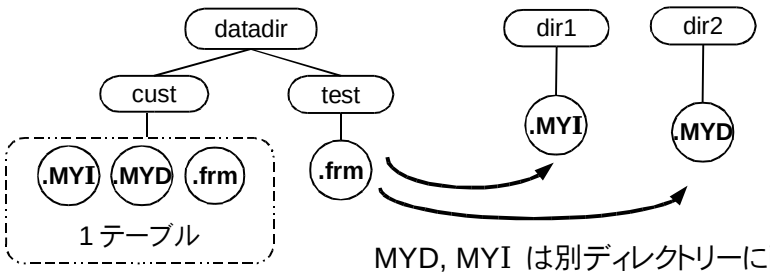
- 違うテーブル型を同時に使用可能。
- JOIN も、テーブル型が違っていても OK 。



32

MyISAM エンジンのファイル構造

- 1 テーブル = 3 つのファイル
- データファイル (MYD) とインデックスファイル (MYI) は他のディレクトリーに置くことが可能。(CREATE TABLE 文で指定できる)
- B-Tree



MYD, MYI は別ディレクトリーに配置できる
SQL 文で指定可能

33

MyISAM テーブルのメンテナンス

- コマンドと、SQL 文で実行可能
- コマンド
  - **myisamchk**
- SQL 文
  - **CHECK TABLE**
  - **ANALYZE TABLE**
  - **OPTIMIZE TABLE**
  - **REPAIR TABLE**

34

ひと休み

今回のあるある

reboot したら
違うマシン

とある SNS アバター集より

35

トランザクション

- 標準では AUTO COMMIT モードが ON
  - **BEGIN (START TRANSACTION)** 文で開始
    - 新しいトランザクションを開始するときは必ず必要
      - でないと直に反映
  - **COMMIT, ROLLBACK, BEGIN** で終了
- **SET AUTOCOMMIT=0;**
  - 全ての文はトランザクション扱い
  - **COMMIT, ROLLBACK, BEGIN** で次の新しいトランザクションが開始

36

auto_increment

- 自動採番
- どこか別のテーブルに覚えているわけではない
- フィールドの定義で指定
  - **INT auto_increment primary key**
- 1 から採番
- MyISAM はファイルに値が記録されており、次の値の決定に使用される
- InnoDB はファイルに記録していない。メモリー内にある。

37

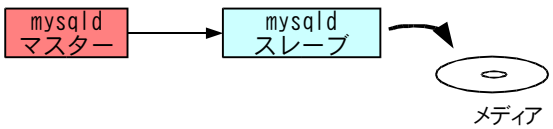
レプリケーション

- 一方向
- 1 つのマスターにつき複数のスレーブ
- 1 つのスレーブは、1 つのマスターしか見れない
- どのストレージエンジンでも動作
- 非同期
  - スレーブのアクション
- 更新はマスターでのみ可能
- レプリケーション対象のデータベースを指定できる
  - 複製したいデータベースだけレプリケーション

38

レプリケーション使用例 1

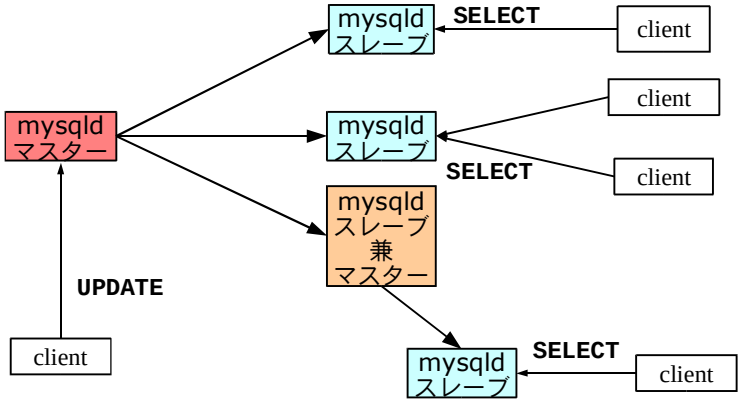
- バックアップ、あるいはスタンバイ目的
  - スレーブサーバーにマスターサーバーのコピーを作成します。
  - スレーブサーバーを常時接続しておかず、ある時間だけ接続して切断するという運用も可能です。
    - バックアップメディアに落とす時の負荷の軽減
    - 世代管理
  - 他のリライアントソフトと組み合わせれば、 Active-Stanby の構成をとって、障害時に自動的に切り替えることが可能です。



39

レプリケーション使用例 2

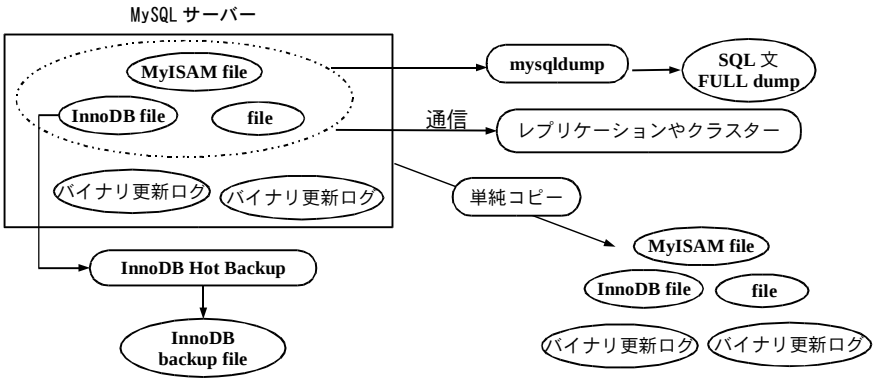
- 負荷の分散
  - スレーブサーバーを大量に作り、検索の負荷を分散させることが可能です。



40

複数のバックアップ方法

- **mysqldump** による SQL 文でのダンプ
- レプリケーションやクラスター
- InnoDB Hot Backup
- バイナリ更新ログファイル (ある時点からの変更点の記録)
- ファイルの単純コピー



41

MySQL のログファイル

- 複数の種類のログ
- ログファイルの書き出し先は自由に指定可能

- バイナリ更新ログ (**--log-bin**)
  - データに対する変更点だけを SQL 文で記録

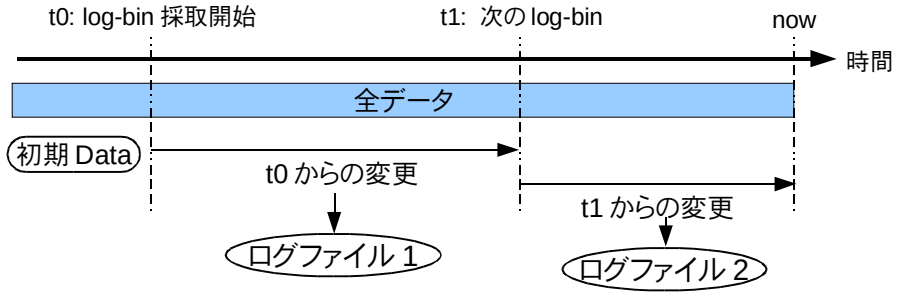
- 詳細ログ (**--log**)
  - いつどのユーザーが接続してきてどのような操作をしたか詳細に記録
    - 接続、切断、ユーザー名、 SELECT 文、とにかく全部記録される
  - テキスト形式
  - セキュリティ監査に使用可能
  - 量がすごい、ローテートしない
    - FIFO ファイルを使う（ログの書き出し先は通常ファイル以外でも可能）
    - 切り出しスクリプト等を書く

- スロークエリーログ (**--log-slow-queries**)
  - 処理に時間のかかったクエリを記録。（**long_query_time=10**）
  - SQL のチューニングに使える

42

バイナリ更新ログファイル

- バイナリ更新ログ (**--log-bin**)
  - 全データに対する変更点だけを SQL 文で記録
  - ログを取りたくないデータベース、テーブルを指定できる
  - データファイルとは違うディレクトリーに書き出せる
  - テキストの SQL 文になる（**mysqlbinlog**）
  - 自動で切り替わる



現時点の全 Data = 初期 Data + ファイル 1 + ファイル 2

43

バイナリ更新ログファイルからのリカバリ

- 手順
  - 初期 Data 用意
  - **mysqlbinlog** コマンドで SQL 文に変換
  - **mysql** コマンドに食わせる

    mysqlbinlog log-bin.000001 | mysql

- **mysqlbinlog** 切り出し範囲指定も可能
  - **--start-datetime, --stop-datetime**
  - **--start-position, --stop-position**

44

Status のモニター

- チューニングするときのヒントとして使われる
- **mysqladmin ext**
  - Status 表示。
-
- **mysqladmin -i 10 ext**
  - 10 秒間隔で永遠に状態をモニターし続ける。
-
- **mysqladmin -i 10 -r ext**
  - 前の Status との差で表示。
-

ひと休み

45

今回のあるある

寝ぼけまなこで
DROP す

とある SNS アバター集より

InnoDB

46

InnoDB 概要

- 行レベルロック
  - ロックエスカレーションはおきない
  - ロックの情報は 1bit/1 レコード しか使用しない
- マルチ・バージョニング
  - 複数のトランザクションが同じレコードを更新、 **SELECT** した場合、 **SELECT** は待たされない
- 4 つのトランザクション分離レベル
- Foreign Key
- Hot Backup ツールの存在
- レプリケーションは InnoDB でも動作。
- なぜかよく聞かれる質問ですが
  - 追記型ではありません。
  - **vacuum** はありません

InnoDB

47

4つのトランザクション分離レベル

- READ UNCOMMITTED, READ COMMITTED, REPEATABLE READ, SERIALIZABLE
- **REPEATABLE READ** が標準 **(** いまのところ **InnoDB** ぐらい **?)**
- **REPEATABLE READ** において、 phantom read は起きない実装
- 変更方法
  - **SET TRANSACTION ISOLATION LEVEL** 文
    - **BEGIN** の前でも後でも実行可能
  - **--transaction-isolation=**
    - システムの標準を変える場合

InnoDB

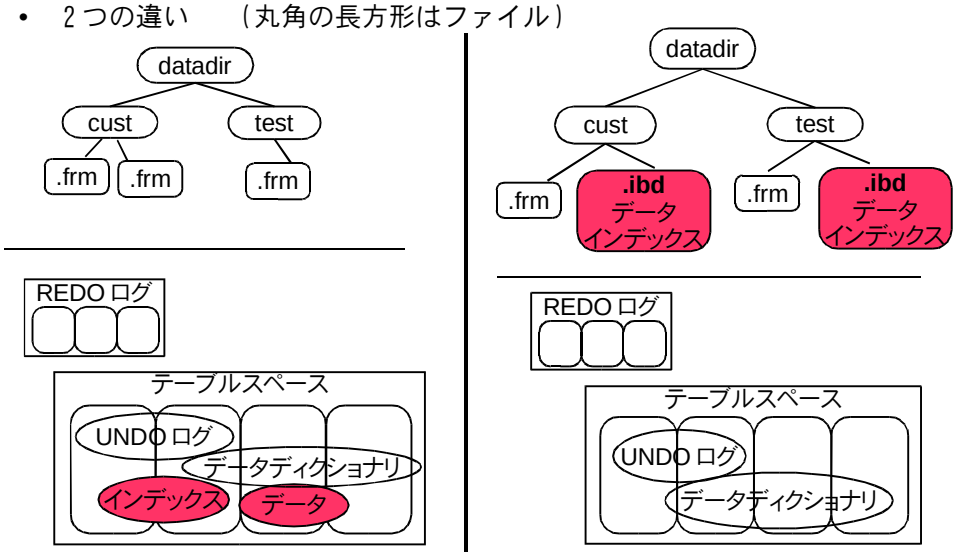
48

InnoDB ファイル構造

- 2 通りの記録方式
  - 1mysqld につき、どちらか一つの方法だけが採用できる。
  - 全ての InnoDB 型のテーブルを同じ領域（テーブルスペース）に保存するか
  - テーブル単位にデータを別のファイルに保存するか (**innodb_file_per_table**)
- テーブルスペース
  - テーブルスペースを構成する複数のファイルは、どのディレクトリに置いてもかまわない。（設定で行う。シンボリック等を使わずとも）
  - それぞれのファイルをばらばらのディレクトリに置いてもかまわない
- REDO ログ
  - REDO ログはデータ（テーブルスペース）とは別のファイルに記録される
  - 循環して利用される。
  - REDO ログファイルを置くディレクトリーは 1 つだけ
  - どのディレクトリーに置いても OK

InnoDB ファイル構造 2

- 2 つの違い　（丸角の長方形はファイル）

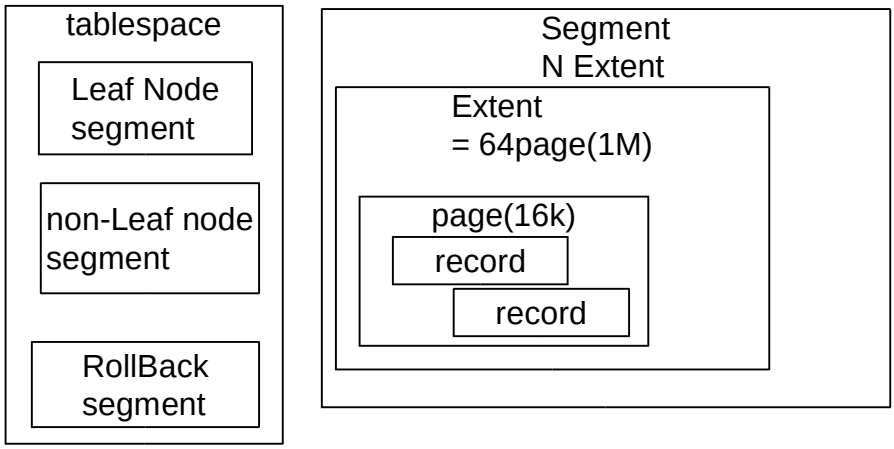


InnoDB のレコードの管理

- 2 つのインデックスで管理
  - レコードはセカンダリー・インデックスとクラスタード・インデックスにより素早く見つけられます。 (B+-Tree)
- クラスタード・インデックス
  - プライマリー・キーに従ってレコードを保存しています。
  - インデックスのリーフはレコードです。
    - といっても、レコードの論理的な並び順に、レコードを物理的に記録するわけではありません。また、新たなレコードの挿入のたびにインデックスを全て作成し直すわけでもありません。
- セカンダリー・インデックス
  - セカンダリー・インデックスは、プライマリー・キーの値をインデックス化しています。

InnoDB の tablespace 構造

- テーブルスペース内には目的に応じた「セグメント」と呼ばれる領域がある
- 各セグメントは、Extent からなる



InnoDB のレコード

- InnoDB は、レコードの値と共に、各種情報をレコードに付加して保存します。
  - Raw ID
  - Transaction ID
  - Roll Pointer
  - 削除フラグ (1bit)
  - 各フィールドポインター
  - 各フィールドの値 (常に可変長)

InnoDB 行ロック

- 1 レコードにつき、 1bit のロック情報しか必要としない
  - それぞれのページに、レコードのロック情報の bitmap がある
  - ある DB は 128 バイトらしいが、それに比べてメモリーの消費が少ない
- 1G メモリーでは、おおよそ 20 億レコードをロックできる
- ロックエスカレーションはない
-
- 注意： InnoDB で行ロックを動作させるには、テーブルに Primary Key が必要です。

ロック競合時の動作 1a

- PG8.0 READ COMMITTED / COMMIT 時

COMMIT時

	A	B	
	test=# BEGIN;	test=# BEGIN;	
	BEGIN	BEGIN	
	test=# SET TRANSACTION ISOLATION LEVEL READ COMMITTED;	test=# SET TRANSACTION ISOLATION LEVEL READ COMMITTED;	
	SET	SET	
更新	test=# UPDATE t SET j=11 WHERE i=1;		
	UPDATE 1		
		test=# UPDATE t SET j=21 WHERE i=1;	
			待たされる
終了	test=# COMMIT;		
	COMMIT		
		UPDATE 1	更新

ロック競合時の動作 1b

- InnoDB READ COMMITTED / COMMIT 時
- 待ち時間が 50″（**innodb_lock_wait_timeout**）になるとデッドロックエラー。

COMMIT時

	A	B	
	mysql> BEGIN;	mysql> BEGIN;	
	Query OK, 0 rows affected (0.00 sec)	Query OK, 0 rows affected (0.00 sec)	
	mysql> SET TRANSACTION ISOLATION LEVEL READ COMMITTED;	mysql> SET TRANSACTION ISOLATION LEVEL READ COMMITTED;	
	Query OK, 0 rows affected (0.00 sec)	Query OK, 0 rows affected (0.00 sec)	
更新	mysql> UPDATE t SET j=11 WHERE i=1;		
	Query OK, 1 row affected (0.00 sec)		
	Rows matched: 1 Changed: 1 Warnings: 0		
		mysql> UPDATE t SET j=21 WHERE i=1;	
			待たされる
終了	mysql> COMMIT;		
	Query OK, 0 rows affected (0.00 sec)	Query OK, 1 row affected (7.61 sec)	更新
		Rows matched: 1 Changed: 1 Warnings: 0	

ロック競合時の動作 2a

- PG8.0 SERIALIZABLE / COMMIT 時
- **READ COMMITTED** の時と動作が変わる。
- **SERIALIZABLE / ROLLBACK** の時とも動作が違う。

COMMIT時

	A	B	
	test=# BEGIN;	test=# BEGIN;	
	test=# SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;	test=# SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;	
更新	test=# UPDATE t SET j=11 WHERE i=1;		
	UPDATE 1		
		test=# UPDATE t SET j=21 WHERE i=1;	
			待たされる
	test=# COMMIT;		
終了	COMMIT		
		ERROR: could not serialize access due to concurrent update	エラー
		test=# SELECT * FROM t;	
		ERROR: current transaction is aborted, commands ignored until end of transaction block	rollback 以外なにもできない
		test=# COMMIT;	打ち間違えた
		ROLLBACK	強制ロールバック

ロック競合時の動作 2b

- InnoDB SERIALIZABLE / COMMIT 時（RC と動作同じ）
- 待ち時間が 50″（**innodb_lock_wait_timeout**）になるとデッドロックエラー。
- InnoDB では、 **(RC | S) | (COMMIT | ROLLBACK)** 時の動作が同じ

COMMIT時

	A	B
	mysql> BEGIN; Query OK, 0 rows affected (0.00 sec)	mysql> BEGIN; Query OK, 0 rows affected (0.00 sec)
	mysql> SET TRANSACTION ISOLATION LEVEL SERIALIZABLE; Query OK, 0 rows affected (0.00 sec)	mysql> SET TRANSACTION ISOLATION LEVEL SERIALIZABLE; Query OK, 0 rows affected (0.00 sec)
更新	mysql> UPDATE t SET j=11 WHERE i=1; Query OK, 1 row affected (0.00 sec) Rows matched: 1 Changed: 1 Warnings: 0	
		mysql> UPDATE t SET j=21 WHERE i=1; 待たされる
終了	mysql> COMMIT; Query OK, 0 rows affected (0.00 sec)	
		Query OK, 1 row affected (8.82 sec) 更新 Rows matched: 1 Changed: 1 Warnings: 0

ロック競合時の動作 3a

- PG8.0 READ COMMITTED / COMMIT 時

COMMIT時

A	B
test=# BEGIN; test=# SET TRANSACTION ISOLATION LEVEL READ COMMITTED;	test=# BEGIN; test=# SET TRANSACTION ISOLATION LEVEL READ COMMITTED;
test=# UPDATE t SET j=11 WHERE i=1;	
OK UPDATE 1	
	test=# UPDATE t SET j=22 WHERE i=2;
	UPDATE 1 OK
test=# UPDATE t SET j=21 WHERE i=2;	
待ち続ける	
	test=# UPDATE t SET j=12 WHERE i=1;
	一息待った後に
	ERROR: deadlock detected Error
	DETAIL: Process 19812 waits for ShareLock on transaction 555; Process 19102 waits for ShareLock on transaction 556; blocked
OK UPDATE 1	
	test=# SELECT * FROM t;
	ERROR: current transaction is aborted, c一切のクエリは拒否される
	test=# ROLLBACK; 明示的にrollback
	ROLLBACK

ロック競合時の動作 3b

- InnoDB READ COMMITTED / COMMIT 時
- デッドロックエラー時は、自動で **ROLLBACK**、トランザクション終了

COMMIT時

	A	B	
	mysql> BEGIN;	mysql> BEGIN;	
	mysql> SET TRANSACTION ISOLATION LEVEL READ COMMITTED;	mysql> SET TRANSACTION ISOLATION LEVEL READ COMMITTED;	
	mysql> UPDATE t SET j=11 WHERE i=1;		
OK	Query OK, 1 row affected (0.00 sec)		
		mysql> UPDATE t SET j=22 WHERE i=2;	
		Query OK, 1 row affected (0.00 sec)	OK
	mysql> UPDATE t SET j=21 WHERE i=2;		
待ちになる			
		mysql> UPDATE t SET j=12 WHERE i=1;	
		ERROR 1213 (40001): Deadlock found when trying to get lock; try restarting transaction	直にdead lock error
Bがエラーで落ちたのでUPDATE	Query OK, 1 row affected (12.60 sec)		
	Rows matched: 1 Changed: 1 Warnings: 0		
		mysql> SELECT * FROM t;	自動でROLLBACKしている
		+---+------+	
		| i | j |	
		+---+------+	
		| 1 | 1 |	

ロック競合時の動作 3c

- InnoDB READ COMMITTED / COMMIT 時
- **InnoDB** はコストベース。先着順ではない。

COMMIT時

	A	B	
	mysql> BEGIN;	mysql> BEGIN;	
	mysql> SET TRANSACTION ISOLATION LEVEL READ COMMITTED;	mysql> SET TRANSACTION ISOLATION LEVEL READ COMMITTED;	
		mysql> UPDATE t SET j=32 WHERE i=3;	更新が1つ多い
		Query OK, 1 row affected (0.00 sec)	OK
	mysql> UPDATE t SET j=11 WHERE i=1;		
OK	Query OK, 1 row affected (0.00 sec)		
		mysql> UPDATE t SET j=22 WHERE i=2;	
		Query OK, 1 row affected (0.00 sec)	OK
	mysql> UPDATE t SET j=21 WHERE i=2;		
待ちになる			
		mysql> UPDATE t SET j=12 WHERE i=1;	
こちらがエラーになる	ERROR 1213 (40001): Deadlock found when trying to get lock; try restarting transaction		
		Query OK, 1 row affected (0.00 sec)	OKになる
		Rows matched: 1 Changed: 1 Warnings: 0	

InnoDB

61

InnoDB のバックアップ

- **mysqldump**
- ファイルのコピー
  - my.cnf ファイル、テーブルスペース、 REDO ログ、 .frm ファイル、 .ibd ファイルをコピーします
- **InnoDB Hot Backup**
  - mysqld を停止する必要はありません
  - InnoDB に関するファイルを読み込んでバックアップファイルを作成します。
  - mysqld を動作させているマシン上で実行します。
  - 有料です (MySQL 本体には含まれておらず、別売されています)

InnoDB

62

InnoDB のフラグメンテーション

- フラグメンテーションは発生しますが、それほど大きな問題にはなりません。
  - セカンダリーインデックスやクラスタードインデックス、内部構造など
  - もしきれいにしたい場合は、
    - 一度 InnoDB のデータ全てを mysqldump で吸い上げて InnoDB 領域を作成し直すか、
    - **ALTER TABLE** 文
    - **OPTIMIZE TABLE** 文
    - InnoDB Hot Backup を使用してバックアップを取り、バックアップから戻す

InnoDB のメンテナンス

- 電源断等の、障害からのリカバリーは自動
- **SQL** 文
  - **CHECK TABLE** 文
  - **ANALYZE TABLE** 文
  - **OPTIMIZE TABLE** 文

InnoDB そのほか

- ロックを明示的に指定したい場合は、
  - **SELECT ... LOCK IN SHARE MODE**
  - **SELECT ... FOR UPDATE**
  - **LOCK TABLES** は使用しない
- BLOB, TEXT は 4G 未満であること
- フィールド数は 1000 まで
- 1 レコードの最大長 (BLOB,TEXT は除く) は、 8KByte まで
- FULLTEXT, GIS は使えない
- mysql データベース以下の権限テーブルを InnoDB 型にしてはいけない
- SELECT COUNT(*) はディスクに記録していない
- auto_increment の値はディスクに記録していない
- Primary Key は必ず張ること

ひと休み

今回のあるある

スロット間違え
RAID とぶ

とある SNS アバター集より

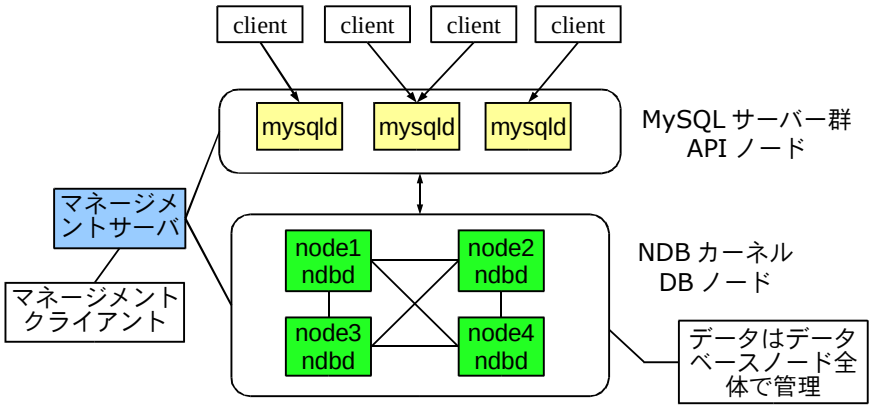
NDB

NDB クラスター概要

- Alzato から MySQL AB に
- MySQL 4.1 のソースに組み込まれています。
-
- データはメモリー内にあります。
  - 高速な動作
  - ログ **(** データベースのイメージ **)** は **disk** に記録されています
- **NDB** 型のテーブル **(** ストレージエンジン **)** 。 **TYPE=ndbcluster**
- **HotBackup** 機能
- 障害時のノードの切り替えは一瞬。
- ノードのメンテナンスは楽。
  - ノードを落としてまたたち上げるだけで復旧。
-
- **NDB** 型は **READ COMMITTED** のみ。
- 行レベルロック

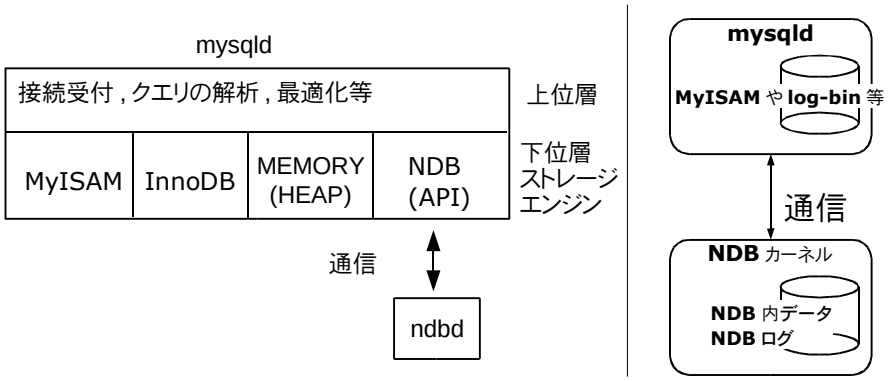
NDB クラスター構造

- 3 層。それぞれの層でスケールアップ可能
- 1DB ノード = 1ndbd
- 1API ノード = 1mysqld(etc...)
- 各ノードはどのハードにインストールしてもかまわない



NDB クラスター構造 2

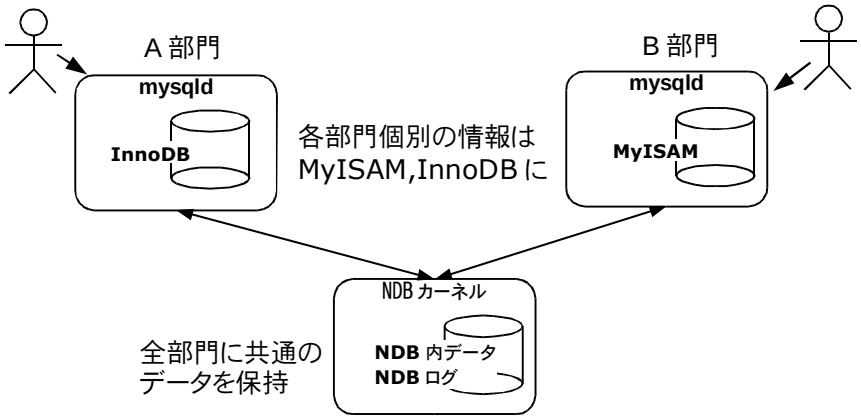
- mysqld にしてみれば、 ndbd は別プロセス
- ndbd にしてみれば、 mysqld は NDB API をもったソフト
- MyISAM,InnoDB は個々の mysqld が管理
- NDB のデータは、全ての mysqld,ndbd で共通



69

NDB クラスターのちょっとした使い方

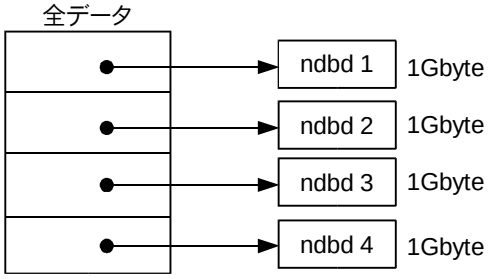
- たんなる冗長性、拡張性、可用性、高速性以外の目的での使い方
- 以下の特徴を逆手にとって、データアクセスの分離
  - MyISAM,InnoDB は個々の mysqld が管理
  - NDB のデータは、全ての mysqld,ndbd で共通



70

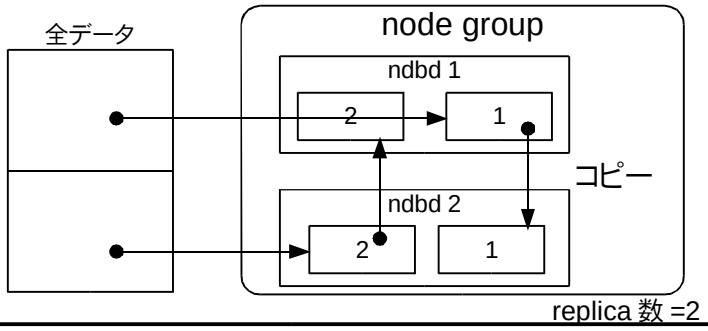
Data Fragment

- 全てのデータ (NDB 型のデータ) は、全ての DB ノードに分割されて保持管理されます。
- メリット
  - 各 ndbd に分割することで、素早い動作が期待できる。
  - ndbd 搭載のマシンを増やすだけで、使えるメモリー量 (保存できるデータ量) が増える
- データベースのイメージログは各 ndbd が動作している機械上に作成されます。



Data Replica

- ある ndbd が管理しているデータは、別の ndbd に複製されます。
  - 安全性が高まります。
  - ndbd が仮に一つおちたとしてもサービスは続けられます。
    - primary と secondary の切り替えは一瞬
  - ndbd のメンテナンス時もサービスが提供可能になります。
- 同じデータを持っている複数の ndbd を、node group と呼びます
- コピーは同期



おわり

何かある?

今回のあるある

ROLLBACK が効きません

とある SNS アバター集より