セルメモリDB (CMDB)

サービス説明書

2012年6月　初版

2014年5月　第2版

(株)トライテック

目　次

[1. 概要 6](#_Toc389036795)

[2. Paxosについて 6](#_Toc389036796)

[3. 高速化について 7](#_Toc389036797)

[3.1. 方針 7](#_Toc389036798)

[4. 構成 8](#_Toc389036799)

[5. 環境設定 8](#_Toc389036800)

[5.1. セルアドレスの取得 8](#_Toc389036801)

[5.2. 単体ポート指定 9](#_Toc389036802)

[5.3. メモリ管理 9](#_Toc389036803)

[5.4. ディレクトリ 9](#_Toc389036804)

[5.5. 日本語 10](#_Toc389036805)

[5.6. ログ 10](#_Toc389036806)

[5.7. Ras管理 11](#_Toc389036807)

[5.8. 遠隔操作 13](#_Toc389036808)

[6. コマンド 14](#_Toc389036809)

[6.1. xjPingPaxos（CMDBサーバ）の起動 14](#_Toc389036810)

[6.2. xjPingPaxosの停止(xjShutdown) 14](#_Toc389036811)

[6.3. xjPingPaxosの情報取得(xjInfo) 15](#_Toc389036812)

[6.4. テーブル一覧(xjList) 15](#_Toc389036813)

[6.5. テーブル情報取得(xjTable) 15](#_Toc389036814)

[6.6. 試験用テーブルの作成(xjCre) 15](#_Toc389036815)

[6.7. テーブルの削除(xjDrop) 16](#_Toc389036816)

[6.8. SQL文の実行(xjSql) 16](#_Toc389036817)

[7. API　（Cライブラリ） 17](#_Toc389036818)

[7.1. データベースとの接続 17](#_Toc389036819)

[7.2. データベースとの切断 17](#_Toc389036820)

[7.3. テーブルの作成 18](#_Toc389036821)

[7.4. テーブルの削除 18](#_Toc389036822)

[7.5. テーブルのオープン 18](#_Toc389036823)

[7.6. テーブルのクローズ 19](#_Toc389036824)

[7.7. 更新データのcommit 19](#_Toc389036825)

[7.8. 更新データのrollback 19](#_Toc389036826)

[7.9. レコードの挿入 20](#_Toc389036827)

[7.10. レコードの削除 20](#_Toc389036828)

[7.11. レコードの更新 21](#_Toc389036829)

[7.12. 検索条件の設定 21](#_Toc389036830)

[7.13. レコードの検索 22](#_Toc389036831)

[7.14. 検索用カーソルの先頭設定 22](#_Toc389036832)

[7.15. テーブルのソート 23](#_Toc389036833)

[7.16. 複数テーブルの排他宣言 23](#_Toc389036834)

[7.17. テーブルの排他解除 23](#_Toc389036835)

[7.18. テーブルの圧縮 24](#_Toc389036836)

[7.19. アイテム値の設定 24](#_Toc389036837)

[7.20. アイテム値の取得 24](#_Toc389036838)

[7.21. レコード長の参照 25](#_Toc389036839)

[7.22. レコード数の参照 25](#_Toc389036840)

[7.23. SQL文の実行 26](#_Toc389036841)

[7.24. SQL文実行の先頭の結果取得 26](#_Toc389036842)

[7.25. SQL文実行の次の結果取得 26](#_Toc389036843)

[7.26. SQL文実行の結果のクローズ 27](#_Toc389036844)

[8. API（Rubyライブラリ） 28](#_Toc389036845)

[8.1. CMDB環境初期化 28](#_Toc389036846)

[8.2. CMDB環境終了化 28](#_Toc389036847)

[8.3. CMDBとの接続 28](#_Toc389036848)

[8.4. テーブル作成 28](#_Toc389036849)

[8.5. テーブルオープン 28](#_Toc389036850)

[8.6. テーブル操作関連 28](#_Toc389036851)

[（１）データ登録 28](#_Toc389036852)

[（２）データ検索＆取得 29](#_Toc389036853)

[（３）SQL文でデータ検索＆取得 (1-SQL文の実行限定） 29](#_Toc389036854)

[（４）データ更新 29](#_Toc389036855)

[（５）レコード削除 (rdb\_search & rdb\_find & rdb\_deleteをまとめて） 29](#_Toc389036856)

[（６）テーブル削除 30](#_Toc389036857)

[（７）検索条件設定 30](#_Toc389036858)

[（８）検索条件データ取得　(cmdb\_serchの後で） 30](#_Toc389036859)

[（９）SQL文の実行試験 (1-SQL文の実行限定） 30](#_Toc389036860)

[8.7. テーブルクローズ 30](#_Toc389036861)

[8.8. CMDBとの切断 30](#_Toc389036862)

[8.9. 【Rubyライブラリ使用例】 31](#_Toc389036863)

[9. API（PHPライブラリ） 33](#_Toc389036864)

[9.1. CMDBとの接続 33](#_Toc389036865)

[9.2. CMDBとの切断 33](#_Toc389036866)

[9.3. テーブル作成 33](#_Toc389036867)

[9.4. テーブル削除 33](#_Toc389036868)

[9.5. テーブルオープン 33](#_Toc389036869)

[9.6. テーブルクローズ 33](#_Toc389036870)

[9.7. DBに対するコミット 34](#_Toc389036871)

[9.8. DBに対するロールバック 34](#_Toc389036872)

[9.9. テーブル操作関連 34](#_Toc389036873)

[（１）データ登録 34](#_Toc389036874)

[（２）データ削除 34](#_Toc389036875)

[（３）データ更新 34](#_Toc389036876)

[（４）検索条件の設定 35](#_Toc389036877)

[（５）レコードデータ取得（search()で設定した結果を得る） 35](#_Toc389036878)

[（６）検索用カーソルを先頭に設定 35](#_Toc389036879)

[（７）テーブルのソート（search()で検索条件にソート条件を追加） 35](#_Toc389036880)

[（８）テーブルの圧縮 36](#_Toc389036881)

[（９）レコード数の参照 36](#_Toc389036882)

[（１０）スキーマ定義参照 36](#_Toc389036883)

[（１１）テーブル名参照 36](#_Toc389036884)

[（１２）カレントテーブルに対するコミット 36](#_Toc389036885)

[（１３）カレントテーブルに対するロールバック 36](#_Toc389036886)

[（１４）SQL文実行 36](#_Toc389036887)

[（１５）SQL文実行の先頭の結果取得 36](#_Toc389036888)

[（１６）SQL文実行の次の結果取得 37](#_Toc389036889)

[（１７）SQL文実行の結果削除 37](#_Toc389036890)

[9.10. 【PHPライブラリ使用例】 37](#_Toc389036891)

[10. SQL文法 39](#_Toc389036892)

[10.1. 型 39](#_Toc389036893)

[10.2. 文 39](#_Toc389036894)

[10.3. schema文 39](#_Toc389036895)

[10.4. select文 40](#_Toc389036896)

[10.5. insert文 41](#_Toc389036897)

[10.6. update文 41](#_Toc389036898)

[10.7. delete文 41](#_Toc389036899)

[10.8. transaction文 41](#_Toc389036900)

[11. 試験 42](#_Toc389036901)

[付録1　NWGadget.TAR、xjPing.TARのbuild方法 43](#_Toc389036902)

# 概要

　セルメモリDBサービス（CMDB）は、高速軽量を目的として組み込み向けに20年前に開発されたメモリRDBMSをベースにクラウド時代用に改修したものである。本来、メモリDBなので高速性は確保されていたが、信頼性に不安があった。即ち、メモリDBのサーバがダウンした場合にはメモリ上にあるキャッシュデータ(故障率をrとすると1‐rの信頼性)が消失してしまう問題である。この問題を当社のセル化による実装で解決し、クラウド時代に向けたメモリDBとして衣替えしたのがCMDBである。

　なお、改造に当たってはSQLに対してwhere句の等値に関するハッシュによる分割統治管理による高速化を行っている。また、SQL言語としては必要最小限を実装している。

　APIとしては、Cライブラリ、PHPライブラリ、Rubyライブラリ、SQLインタプリターを用意してある。

# Paxosについて

キャッシュデータの信頼性は、従来の現行系及び待機系の2重化構成では1‐r\*\*2である。一方Paxosプロトコルを使用したサーバのセル化（n構成）では信頼性は1‐r\*\*nとなる。このようにセル化では台数を増やすことにより「べき乗」で信頼性を高めることができる。例えば、信頼性がナイン9 (99.9999999%)以上であれば永続化は必要ないとすることができる。

性能については、キャッシュデータの高信頼性により、コストを要するPaxosインスタンスのログのディスク書出しのアトミックな永続化を省くことができ飛躍的な高速化を図ることができる。

また、従来の現行系/待機系の2重化システムでは現行系は待機系の実行完了を待つのが基本であったが、当社のセル化は合意完了後の実行が突き放され他のサーバの実行完了を待つ必要がない。実行完了過程を省けるのは、合意に参加したが実行未完であれば、次の合意に参加できずPaxos合意過程から離脱することになり、そして参加者が過半数を割れば進行が停止することが保証されているからである。

また、実行完了を待つ場合には、各サーバは並列動作するにしてもサーバ毎の特性及び通信特性等によりわずかな遅延が発生し影響を及ぼすことになるが、当社のセル化方式では突き放しなので実行完了応答待ちの遅延がなく高速化が実現できる。

なお、参照については同期を強制しない場合は、分散参照が行われ複数クライアントの参照は並列化され、クライアント多重化が行われる。

# 高速化について

## 方針

　高速化についての方策を低レベルから記述する。

1. **メモリ**

テーブルデータに対する操作は、全データをメモリ上にロードし、処理を行っている。従って、ロード時以外のコストは純粋にアルゴリズムのオーダー(O)に依存する。

Cライブラリ、PHPライブラリ、Rubyライブラリのテーブル操作は、１実体テーブルに対する操作とし、全件(n)処理としている。即ち、O(n)である(インデックス機能を付加すれば、O(log(n))を期待できる)。

複数テーブルに対する操作は、SQLによる。

1. **等値ハッシュによる分割統治管理**

SQLのwhere句の等値に着目し、等値に合致するレコード群のみについて処理を行う分割統治を行っている。等値ハッシュのサイズ、ハッシュ関数は全てに共通としているので、同一ハッシュ値でのみで処理を行えばよい。即ち、where句に一つでも等値があれば等値の部分集合について処理を行い、高速化を行っている。

等値のハッシュサイズをs、平均ハッシュ長さをkとすれば、

アイテム=アイテムの場合O(s\*k\*k)=O(n\*k)であり、全件操作のO(n\*n)のk/sのコストとなる。

アイテム=定数の場合O(k)であり、全件操作のO(n)のk/sのコストとなる。

なお、CMDBはkが一定数を超える場合、2次ハッシュを使用しているので、最大k/(s1\*s2)のコストとなる。（一定数:200、s1:1009、s2:101としている）

1. **ビュー**

ビューとは、複数テーブルから作成される参照用の仮想テーブルである。一度、作成されれば毎回作成する必要がないので高速化を図ることができる。

なお、CMDBは、ビューについては時間の依存関係を認識し元テーブルが更新されたならば自動的にビューを作り直す。

1. **union　allによる分割統治**

大容量データの操作の高速化の基本は、分割統治原理である。即ち、大容量データに対して必要な操作を分割統治により処理を減らすのが基本である。

このため、CMDBでサポートしているSQL言語は簡易仕様であるが、アプリケーションの分割統治のために「union all」を用意している。「union all」は、テーブルを横に分割し高速化を図る。

# 構成

xjPingPaxos　0

xjPingPaxos　1

xjPingPaxos　2

Application

セル使用向け

ライブラリ

単体使用向け

ライブラリ

コマンド(セル向け)

xjCre

xjDrop

xjSql

・・・

コマンド(単体向け)

xjShutdown

xjInfo

xjCre

xjDrop

xjSql

・・・

CMDB

# 環境設定

## セルアドレスの取得

　セル名(CellName)に対して、次のファイル名で

${HOME}/\_CellName.conf　　　　あるいは

/etc/CellName/server.conf

に作成する。

　ファイルの内容は、以下のようである。

#id hostname UDP TCP bin DB root

1. paxos03 4100 4110 ~/bin 4120 ~/0
2. paxos04 4101 4111 ~/bin 4121 ~/1
3. paxos05 4102 4112 ~/bin 4122 ~/2
4. …

　id（セルを構成するノード識別ID）からホスト名を参照し、ホスト名から/etc/hostsあるいはDNSでipアドレスを取得する。UDPはセル用のポート番号、TCPはセッション用のポート番号、binはロードモジュールの配布先である。

　CMDBでは、さらに

　DB 単体/管理用ポート番号

　root データベースroot

を指定する。

なお、CMDBでは、

　$bin/paxosにセル用コマンド

　$bin/cmdに単体用コマンド

を作成する。

## 単体ポート指定

　CMDBは、単体でも動作する仕様となっている。また、本ポートは管理ポートとしても使用される。

単体ポートは

環境変数=ipアドレス：ポート番号

で指定する。たとえば、

export　H\_0=192.168.222.3:5100

とする。

## メモリ管理

　環境変数　PING\_MEM\_MAXに最大バイト数を設定するとCMDB自前のメモリ管理が使用される。

PING\_MEM\_MAX=最大バイト数[/Best]

最大バイト数は、メガバイト単位

‘/Best’が指定されるとB木によるベストフィット方式が選ばれ、指定がないとリストによるファーストフィット方式が選択される。

　環境変数が指定されていなければ、システムのメモリ管理(malloc/free)が選択される。

## ディレクトリ

　同一マシン上での稼働の場合である。（遠隔時は、deploy-xjping-conf.shによる）

　InitDB.sh　ディレクトリ

でディレクトリを作成する。

例えば、InitDB.sh　1とすると、

1

1/DB データベース

1/DATA 大容量データ

が作成される。

## 日本語

　日本語コードは、EUC-JPを使用している。

　「xjSql」は、コマンドラインでEUCコードを指定するのは難易度が高いのでシェルスクリプトとして変換されたファイルとするのがよい。

　シェルスクリプトの先頭に、

#!/bin/sh

LANG=EUC-JP

を宣言する。

　また、realineライブラリを使用しているので、「.inputrc」に

set convert-meta off  
set input-meta on  
set output-meta on

を設定する。

## ログ

　環境変数　LOG\_SIZEを指定する。stderrに出力される。

　LOG\_SIZE=0とすると、直接書式変換され、出力される。

　LOG\_SIZEに0以外を指定すると、LOG\_SIZEのバッファを取得し、書式変換なしに、バッファに書き出される。バッファが一杯になると、バッファ全体をそのまま出力する。書式変換出力は、LogPrintによる。

　また、ログファイルを複数ファイルに保存する場合は、LogFilesによる。

コマンド：

LogPrint　[ファイル]

説明:

ファイルが指定されていればファイルから、指定されていなければ標準入力からバッファを読み出し、書式変換出力を標準出力に行う。

例：

LOG\_SIZE=10000　xjPingPaxos　…　|　LogPrint

10Kバイト毎にログ出力を行う。

コマンド：

LogFiles　ファイル　最大ファイル数

説明：

標準入力からバッファを読み出しバッファ毎にファイルを最大ファイル数個作成する。

ファイル名は、「ファイル－xxxx」の通番で最大ファイル数個保存される。

例：

LOG\_SIZE=10000　xjPingPaxos　…　|　LogFiles　LOG　10

10Kバイト毎に、LOG-0000、LOG-0001、LOG-0002、・・・とログファイルが作成される。

ログ内容の表示は、LogPrint　LOG-000　とする。

## Ras管理

　Ras管理セルを使用する場合には、環境変数　RAS\_CELL　にセル名を設定する。

セル名が設定されていればRas管理セルに本セル名のディレクトリにid名の一時ファイルを作成し、本ディレクトリに対するイベント待ちを行う。

(1) RAS管理の例

Cell Storage Service (CSS)

(RASセル)

AP

RAS用ｽﾚｯﾄﾞ

AP

RAS用ｽﾚｯﾄﾞ

AP

RAS用ｽﾚｯﾄﾞ

ﾏｽﾀｰ候補通知

障害発生

障害検知

障害検知

登録

登録

RAS通知

RAS通知

ﾏｽﾀｰ候補通知

　上図の場合、アプリケーションと一体のRAS用スレッドを用意し、CSS(Cell　Storage　Service)にエフェメラルファイルを登録し、ディレクトリに対するイベント取得を登録する。

障害が発生したら直ちにイベントがRAS用スレッドに送られ、RAS用スレッドはRAS通知を自身に送る。Ras通知後、アプリケーションのPaxosモジュールはマスター候補を選び直し、他のサーバに送信する。これにより、マスター交替が瞬時に行われる。

(2) 実装方法

1. セッションのオープン

PaxosCellGetAddr(Rasセル名、&RasSessionCell、NULL)

でセルアドレスを取得し、PaxosSessionInit()、PaxosSessionOpen()でRasセルに接続する。

1. 登録

PFSRasRegister(pRasSession、ディレクトリ名、ファイル名)

で登録すると、ディレクトリ名／ファイル名でエフェメラルファイルが作成される。

1. ダウンイベント取得

PFSRasThreadCreate(pRasSession、ディレクトリ名、&PaxosCell、MyId、NULL)

とすると、ディレクトリ名でのイベントを設定し、自身にダウンを通知する。NULLにint(\*fCallback)(struct　Session\*、PFSHead\_t\*)を設定すると、イベントを取得できる。処理後、PFSRasSend(struct　Session、PFSHead\_t\*)を呼び出すと自身にダウンが通知される。

(3) Ras関連コマンド

PFSRasActive　Rasセル名　ディレクトリ　エフェメラルファイル名

説明：

　アプリケーションにRas機能が装備されていない場合に、本コマンドプロセスを起動する。Rasセルのディレクトリにエフェメラルファイルを作成し、無限待ちとなる。システムダウン等で本コマンドプロセスの消失がRasセルで検知されれば、エフェメラルファイルは削除される。

PFSRasMonitor　Rasセル名　ディレクトリ１　ディレクトリ２　・・・

説明：

　Rasセルのディレクトリイベントを監視する。ディレクトリ下のエントリの生成、消滅、更新のイベントが報告される。

　これにより、機器監視ができる。

## 遠隔操作

　構成ファイルをもとに遠隔操作を行う。

事前準備として、ssh系コマンドを使用するので、認証環境を設定しておく。

1. ロードモジュールの配備

コマンド：

deploy-xjping-bin.sh　構成ファイル　id

説明:

　構成ファイルを参照し、idと一致するホストに対し、指定ディレクトリ下にロードモジュールを転送する。

単体コマンドは、$bin/cmd

セルコマンドは、$bin/paxos

に転送する。

1. 構成ファイルの配備と初期化

コマンド：

deploy-xjping-conf.sh　構成ファイル　id

説明:

　構成ファイルを参照し、idと一致するホストに対し、構成ファイルをホームディレクトリ下に転送し、データ用のディレクトリを作成する。

$root/DBはデータベース用、$root/DATAはPaxos帯域外データ（大容量データ）用である。

1. サーバ起動

コマンド：

ssh-start.sh　セル名　id　{simple|paxos　[autonomic　[target]]}

説明:

セル名から構成ファイルを参照し、idと一致するホスト上で、$bin/paxos/start-xjPingPaxos.shを起動する。

* セル名とidのみの場合は、単体サーバとなる。
* paxosを指定すればセル構成となる。
* autonomicのみの指定は過半数のサーバが稼働している条件でマスターサーバから復旧する。
* targetを指定すればtargetから復旧する。
* start-xjPingPaxos.shはロードモジュールの配備で転送されている。
* start-xjPingPaxos.shは、セル名から構成ファイルを参照し、hostnameから自身のid、Cellポート番号、sessionポート番号、管理ポート番号、rootディレクトリ及びロードモジュールディレクトリを取得し、xjPingPaxos（CMDBサーバ）を起動する。
* また、環境変数H\_$id=$HOST:$controlを設定する。paxosが指定されていればRAS\_CELL=rasを設定する。

# コマンド

## xjPingPaxos（CMDBサーバ）の起動

　各サーバの実行ディレクトリで起動する。

コマンド：

xjPingPaxos　ポート環境変数　ディレクトリ　[セル名　id　[autonomic　target\_id]]

例：

xjPingPaxos　H\_1　1　xjPingPaxos　1　autonomic　0

* xjPingPaxosは、単体及びセル構成の両方で動作する。
* 「ポート環境変数」は単体用及び管理コマンド用のポートに使用される。
* 「ディレクトリ」はDB用のルートディレクトリである。
* 「ポート環境変数」と「ディレクトリ」のみを指定すると単体での動作となる。
* 「セル名」と「id」を指定するとセル構成となる。
* さらに、「autonomic」と「target\_id」を指定すると、既に動作している「target\_id」のサーバから状態を取得し、セルに参入する。

## xjPingPaxosの停止(xjShutdown)

　サーバ毎に停止することができる。

コマンド：

xjShutdown　ポート環境変数

例：

xjShutdown　H\_1

Ras管理機能を動作させていない場合、停止時に他のサーバにRASコマンドを送信することで、瞬時にマスター交替を行うことができる。

例：

PaxosAdmin　xjPingPaxos　0　ras　1

PaxosAdmin　xjPingPaxos　2　ras　1

PaxosAdmin　xjPingPaxos　3　ras　1

・・・

セル名がxjPingPaxosの0,2,3に1がダウンしたことを通知する。

## xjPingPaxosの情報取得(xjInfo)

サーバ毎の情報を取得できる。

コマンド：

xjInfo　ポート環境変数

例：

xjInfo　H\_1

## テーブル一覧(xjList)

サーバ毎あるいはセルの情報を取得できる。

コマンド：

xjList(単体用)　ポート環境変数

xjList(セル用)　セル名

例：

xjList　H\_1

## テーブル情報取得(xjTable)

テーブル毎の情報を取得できる。

コマンド：

xjTable　ポート環境変数　テーブル名　[開始レコード　終了レコード]

例：

xjTable　H\_1　B1　0　10

## 試験用テーブルの作成(xjCre)

サーバ毎あるいはセルに試験用テーブルを作成する。

コマンド：

xjCre(単体用)　ポート環境変数　テーブル名　[テーブルの数　レコードの数]

xjCre(セル用)　セル名　テーブル名　[テーブルの数　レコードの数]

例：

xjCre　H\_1　B1　1　100

## テーブルの削除(xjDrop)

サーバ毎あるいはセルからテーブルを削除する。

コマンド：

xjDrop(単体用)　ポート環境変数　テーブル名

xjDrop(セル用)　セル名　テーブル名

例：

xjDrop　H\_1　B1

## SQL文の実行(xjSql)

SQL文を標準入力から読み込み、実行結果を標準出力に書き出す。（SQL文法を参照）

コマンド：

xjSql(単体用)　ポート環境変数

xjSql(セル用)　セル名

例：

xjSql　H\_1

# API　（Cライブラリ）

　単体用にはlibneo.a、セル用にはlibneo2.aを使用する。因みに、libneoa1.aはサーバー用である。

　ヘッダーはneo\_db.hをインクルードする。

　特に、Sql文の実行は次のようにする。

pMd=rdb\_link(“CLIENT”,　ポート環境変数名/セル名);

SQL文の設定

SqlExecute(pMd,　“SQL文”);

SqlResultFirstOpen(pMd,　&pTd);

pTd!=NULLであればレコード参照

SqlResultClose(pTd);

rdb\_unlink(pMd);

## データベースとの接続

[形式]

#include　“neo\_db.h”

**r\_man\_t\***

**rdb\_link(p\_name\_t　MyName,　r\_man\_name\_t　Database)**

[機能]

　MyNameには”CLIENT”を指定する。Databaseにはポート環境変数名(単体)あるいはセル名(セル)を指定する。

[返却値]

　異常時は、NULLが返却される。

## データベースとの切断

[形式]

#include　“neo\_db.h”

**int**

**rdb\_unlink(r\_man\_t\*　pMd)**

[機能]

　pMdで指定されたデータベースと切断する。

[返却値]

　異常時は、-1が返却される。

## テーブルの作成

[形式]

#include　“neo\_db.h”

**r\_tbl\_t\***

**rdb\_create(r\_man\_t\*　pMd,　r\_tbl\_name\_t　name,　int　n,　r\_item\_t　items[n])**

[機能]

　pMdで指定されたデータベースに、nameで指定された名前のテーブルを、totalで指定された初期レコード数及びitems[n]で指定されたスキーマで作成する。

　スキーマには、アイテム名、属性及び有効長を指定する。有効長はsizeof(int)に切り上げられて領域が確保される。

　insert時にレコード領域が不足すれば、レコード数及びレコード領域は自動的に倍で拡張される。delete時には領域は削除されない。圧縮はrdb\_compress()による。

[返却値]

　正常時にはテーブル記述子が返却され、異常時はNULLが返却される。

## テーブルの削除

[形式]

#include　“neo\_db.h”

**int**

**rdb\_drop(r\_man\_t\*　pMd,　r\_tbl\_name\_t　name])**

[機能]

　pMdで指定されたデータベースから、nameで指定された名前のテーブルを削除する。

オープン中であればE\_RDB\_BUSYのエラーとなる。

[返却値]

　正常時には0が返却され、異常時は-1が返却される。

## テーブルのオープン

[形式]

#include　“neo\_db.h”

**r\_tbl\_t\***

**rdb\_open(r\_man\_t\*　pMd,　r\_tbl\_name\_t　name)**

[機能]

　pMdで指定されたデータベースのnameで指定された名前のテーブルをオープンする。

[返却値]

　正常時にはテーブル記述子が返却され、異常時はNULLが返却される。

## テーブルのクローズ

[形式]

#include　“neo\_db.h”

**int**

**rdb\_close(r\_tbl\_t\*　pTd)**

**rdb\_close\_client(r\_tbl\_t\*　pTd)**

[機能]

　pTdで指定されたテーブルをクローズする。

　rdb\_closeはクローズ時に更新データはcommitされる。

rdb\_close\_clientはクライアントの領域を解放するだけであり、更新データの反映の有無はSQL文のcommit/rollbackによる。

[返却値]

　正常時には0が返却され、異常時は-1が返却される。

## 更新データのcommit

[形式]

#include　“neo\_db.h”

**int**

**rdb\_flush(r\_tbl\_t\*　pTd)**

[機能]

　pTdで指定されたテーブルの更新データをcommitする。

[返却値]

　正常時には0が返却され、異常時は-1が返却される。

## 更新データのrollback

[形式]

#include　“neo\_db.h”

**int**

**rdb\_cancel(r\_tbl\_t\*　pTd)**

[機能]

　pTdで指定されたテーブルの更新データをrollbackする。

[返却値]

　正常時には0が返却され、異常時は-1が返却される。

## レコードの挿入

[形式]

#include　“neo\_db.h”

**int**

**rdb\_insert(r\_tbl\_t\*　pTd,　int　n,　char　\*rec,**

**int　m,　r\_item\_nam\_t　item[m],　int　mode)**

[機能]

　pTdで指定されたテーブルに、recで指定された連続したnレコードのデータを挿入する。挿入時には空き領域を探し順次設定する。空き領域がない場合には自動拡張が行われる。

　mとitem[m]が指定されている場合には指定カラムについての重複チェックを行う。これによりキーチェックを行うことができる。

　modeにはR\_EXCLUSIVEあるいはR\_SHAREを指定する。

[返却値]

　正常時には0が返却され、異常時は-1が返却される。

[制約]

　重複チェックは、１レコード挿入時のみである。

## レコードの削除

[形式]

#include　“neo\_db.h”

**int**

**rdb\_delete(r\_tbl\_t\*　pTd,　int　n,　char　\*rec,　int　mode)**

[機能]

　pTdで指定されたテーブルに、recで指定された連続したnレコードのデータを削除する。

　レコード待ちのクライアントには、E\_RDB\_NOEXISTのエラーを返す。

　modeにはR\_WAITあるいはR\_NOWAITを指定する。

[返却値]

　正常時には0が返却され、異常時は-1が返却される。

[制約]

　現版では、nは1のみである。

## レコードの更新

[形式]

#include　“neo\_db.h”

**int**

**rdb\_update(r\_tbl\_t\*　pTd,　int　n,　char　\*rec,　int　mode)**

[機能]

　pTdで指定されたテーブルに、recで指定された連続したnレコードのデータを更新する。

　modeにはR\_WAITあるいはR\_NOWAITを指定する。

[返却値]

　正常時には0が返却され、異常時は-1が返却される。

[制約]

　現版では、nは1のみである。

## 検索条件の設定

[形式]

#include　“neo\_db.h”

**int**

**rdb\_search(r\_tbl\_t\*　pTd,　char\*　str)**

[機能]

　pTdで指定されたテーブルに、strで指定された検索条件を設定する。設定後、rdb\_find()で取得する。

　検索条件は文字列であり、左辺値にカラム名を記述する。条件には、比較演算(==,!=,<,>,<=,>=)及び論理演算(&&,||)を記述できる。文字列は、’で囲む。また、()で評価順序を指定できる。

　また、strをNULLとすると全レコードが対象となる。

[返却値]

　正常時には0が返却され、異常時は-1が返却される。

## レコードの検索

[形式]

#include　“neo\_db.h”

**int**

**rdb\_find(r\_tbl\_t\*　pTd,　int　n,　char\*　rec,　int　mode)**

[機能]

　pTdで指定されたテーブルから、rdb\_search()で設定された検索条件で、最大nレコードをrecを設定する。

　modeには、R\_EXCLUSIVEあるいはR\_SHAREを指定する。

　また、R\_WAITを排他中の待ちを指定できる。指定されていない場合に排他されている時はE\_RDB\_BUSYエラーとなる。

　返却値が0となるまで検索する必要がある。

　検索毎に検索用カーソルが進められる。

[返却値]

　正常時には取得レコード数が返却され、異常時は-1が返却される。

## 検索用カーソルの先頭設定

[形式]

#include　“neo\_db.h”

**int**

**rdb\_rewind(r\_tbl\_t\*　pTd)**

[機能]

　pTdで指定されたテーブルの検索用カーソルを先頭に設定する。

[返却値]

　正常時には0が返却され、異常時は-1が返却される。

## テーブルのソート

[形式]

#include　“neo\_db.h”

**int**

**rdb\_sort(r\_tbl\_t\*　pTd,　int　n,　r\_sort\_t　sort[n])**

[機能]

　pTdで指定されたテーブルを、sortで指定されたn個のキーで、検索条件の範囲でソートする。

　sortにはアイテム名と昇順(R\_ASCENDANT)あるいは降順(R\_DESCENDANT)を指定する。

[返却値]

　正常時には0が返却され、異常時は-1が返却される。

## 複数テーブルの排他宣言

[形式]

#include　“neo\_db.h”

**int**

**rdb\_hold(int　n,　r\_hold\_t　hds[n],　int　wait)**

[機能]

　hdsで指定されたn個のテーブルに排他を宣言する。

　排他モードには、R\_EXCLUSIVEあるいはR\_SHAREを設定する。

　waitにR\_WAITを設定すると解除されるまで待つ。

[返却値]

　正常時には0が返却され、異常時は-1が返却される。

## テーブルの排他解除

[形式]

#include　“neo\_db.h”

**int**

**rdb\_release(r\_tbl\_t\*　pTd)**

[機能]

　pTdで指定されたテーブルの排他を解除する。

　待ちクライアントが居れば解除を通知する。

[返却値]

　正常時には0が返却され、異常時は-1が返却される。

## テーブルの圧縮

[形式]

#include　“neo\_db.h”

**int**

**rdb\_compress(r\_tbl\_t\*　pTd)**

[機能]

　pTdで指定されたテーブルを圧縮する。他クライアントが使用していれば圧縮を行わない。

[返却値]

　正常時には0が返却され、異常時は-1が返却される。

## アイテム値の設定

[形式]

#include　“neo\_db.h”

**int**

**rdb\_set(r\_tbl\_t\*　pTd,　char　\*rec,　r\_item\_name\_t　item\_name,　void\*　data)**

[機能]

　pTdで指定されたテーブルレコードrecにitem名のオフセットと長さでdataを設定する。

dataは、対象itemのサイズ分の領域が確保されていること。設定時にitemのサイズ分を無条件にrecに設定する。itemがCHAR型の場合itemサイズより短い内容を設定する場合にもitemのサイズ分の領域確保、設定内容より残り部分の0クリアをする必要がある。

[返却値]

　正常時には0が返却され、異常時は-1が返却される。

## アイテム値の取得

[形式]

#include　“neo\_db.h”

**int**

**rdb\_get(r\_tbl\_t\*　pTd,　char　\*rec,　r\_item\_t　item,　void\*　data,　int　\*len)**

[機能]

　pTdで指定されたテーブルからitemのオフセットと長さでrecからdataに設定する。

　\*lenには有効長を設定する。

[返却値]

　正常時には0が返却され、異常時は-1が返却される。

## レコード長の参照

[形式]

#include　“neo\_db.h”

**int**

**rdb\_size(r\_tbl\_t\*　pTd)**

[機能]

　pTdで指定されたテーブルのレコードの長さを参照する。

[返却値]

　正常時にはレコード長が返却され、異常時は-1が返却される。

## レコード数の参照

[形式]

#include　“neo\_db.h”

**int**

**rdb\_number(r\_tbl\_t\*　pTd,　int　mode)**

[機能]

　pTdで指定されたテーブルのレコード数を参照する。

　modeには以下を指定する。

1. 総レコード数
2. 使用中レコード数
3. 未使用レコード数

[返却値]

　正常時にはレコード数が返却され、異常時は-1が返却される。

## SQL文の実行

[形式]

#include　“neo\_db.h”

**int**

**SqlExecute(r\_man\_t\*　pMd,　char\*　statement)**

[機能]

　pMdで指定されたデータベースに対し、statementで指定されたSQL文を実行する。

　実行結果は、pMdの結果リストに記憶される。

[返却値]

　正常時には0が返却され、異常時は-1が返却される。

## SQL文実行の先頭の結果取得

[形式]

#include　“neo\_db.h”

**int**

**SqlResultFirstOpen(r\_man\_t\*　pMd,　r\_tbl\_t\*\*　ppTd)**

[機能]

　pMdで指定されたデータベースの結果リストの先頭を\*ppTdに設定し、レコードをサーバから取得する。

　\*ppTdがNULLであれば終了である。

[返却値]

　正常時には0が返却され、異常時は-1が返却される。

## SQL文実行の次の結果取得

[形式]

#include　“neo\_db.h”

**int**

**SqlResultNextOpen(r\_man\_t\*　pMd,　r\_tbl\_t\*\*　ppTd)**

[機能]

　pMdで指定されたデータベースの次の結果を\*ppTdに設定し、レコードをサーバから取得する。

　\*ppTdがNULLであれば終了である。

[返却値]

　正常時には0が返却され、異常時は-1が返却される。

## SQL文実行の結果のクローズ

[形式]

#include　“neo\_db.h”

**int**

**SqlResultClose(r\_tbl\_t\*　pTd)**

[機能]

　pTdで指定されたテーブルをクローズする。

　また、結果リストからも削除される。

[返却値]

　正常時には0が返却され、異常時は-1が返却される。

**【SQL文実行の使用例】**

if( SqlExecute( pmd, “select \* from tableA;” ) ) {

return; //error処理）

}

while( SqlResultFirstOpen( md, &td ) == 0 && td ) {

// SQL実行結果データの参照例（実行結果は、テーブル定義と同じI/F）

for(i=0,rp = (char\*)td->t\_rec; i < td->t\_rec\_num; i++,rp += td->t\_rec\_size) {

print\_record( i, (r\_head\_t\*)rp, td->t\_scm->s\_n, td->t\_scm->s\_item );

// s\_n, s\_itemの情報をもとにitemデータを参照

}

printf("Total=%d\n", td->t\_rec\_num );

// 参照済みで不要になった結果データの廃棄

SqlResultClose( td );

}

# API（Rubyライブラリ）

【Rubyライブラリmethodの使用例】

require "./CmdbAccess.so"

ttt = CmdbAccess.new

ttt.cmdb\_prologue("myid01")

下記での記法：

　Array(Hash) ： (カラム名,値)のHashオブジェクトからなる配列を意味する

## CMDB環境初期化

**boolean cmdb\_prologue(myid)**

myid: client識別文字列。

return値= falseの時エラー

## CMDB環境終了化

**boolean cmdb\_epilogue()**

return値= falseの時エラー

## CMDBとの接続

**boolean cmdb\_link(myname, cellname)**

myname: client識別文字列。複数クライアント同時接続時の区別用。

(ex. "CLIENT")

cellname: CELL名 or ipaddr:port (ex. 192.168.222.1:2000)

return値= falseの時エラー

## テーブル作成

**boolean cmdb\_create(tablename, schema)**

tablename: テーブル名

schema: レコード定義スキーマ

　(ex. "col1 int,col2 long,col3 char(12),col4 ulong")

※可能な型: int, uint, long, ulong, number(=ulong), float, char,

varchar, date(=long)

　注意：カラム "id ulong"は、内部処理で自動追加される。

カラム名:id は使用禁止。

return値= falseの時エラー

## テーブルオープン

**boolean cmdb\_open(tablename)**

tablename: テーブル名

return値= falseの時エラー

## テーブル操作関連

### （１）データ登録

**Array(Hash) cmdb\_insert(tablename, array\_data, isshare)**

tablename: テーブル名

array\_data: Array(Hash)、登録データ

指定のないカラムは,insert内容を NULL/0 or 0.0とする。

idは、自動生成してinsert時のidに設定。

array\_dataが長さゼロの配列の場合は、エラー。

isshare: 排他モード !=0 :SHARE =0 : EXCLUSIVE (現在、ダミー）

return値= nilの時エラー

Array(Hash): key=カラム名、val=値（type対応のruby Object)

### （２）データ検索＆取得

**Array(Hash) cmdb\_select(tablename, condition, isshare)**

tablename: テーブル名

condition: 検索条件文字列。

左辺にカラム名

比較演算： ==, !=, <, >, <=, >=

論理演算： &&, ||

条件値の文字列は ' で括る

NULLの場合は全レコードが対象

Ex. "id == 123", "id > 123", "NAME == 'PAXOS S3'"

"(id > 123) && (NAME == 'PAXOS S3')" (評価順は左側の()から）

isshare: 排他モード !=0 :SHARE =0 : EXCLUSIVE (現在、ダミー）

return値= nilの時エラー

Array(Hash): key=カラム名、val=値（type対応のruby Object)

検索データが1件もない場合は、長さゼロの配列が返される。

### （３）SQL文でデータ検索＆取得 (1-SQL文の実行限定）

**Array(Hash) cmdb\_sqlselect(sqlstatement, maxrecord)**

sqlstatement: sqlコマンド文字列

maxrecord: 最大レコード数

return値= nilの時エラー

Array(Hash): key=カラム名、val=値（type対応のruby Object)

検索データが1件もない場合は、長さゼロの配列が返される。

【sqlstatement定義例】

"select \* from s3\_object\_data where bucket = 2 and name like 'o%'

order by name"

return---> s3\_object\_data の全カラムのデータ

"select s3\_object\_data.id, s3\_object\_data.name from s3\_object\_data

　　　where bucket = 2 and name like 'o%' order by name"

return---> s3\_object\_data のカラムid, name のデータ

### （４）データ更新

**boolean cmdb\_update(tablename, array\_data, isnowait)**

tablename: テーブル名

array\_data: Array(Hash)、更新データ

Array(Hash): key=カラム名、val=値（type対応のruby Object)

全カラム定義。

内部処理でidでレコードを特定している。

isnowait: 排他モード !=0 :NOWAIT =0 : WAIT

(現在、排他は未サポート→ =1で使用）

return値= falseの時エラー

### （５）レコード削除 (rdb\_search & rdb\_find & rdb\_deleteをまとめて）

**boolean cmdb\_delete(tablename, condition, isnowait)**

tablename: テーブル名

condition: 検索条件文字列。

左辺にカラム名

比較演算： ==, !=, <, >, <=, >=

論理演算： &&, ||

条件値の文字列は ' で括る

Ex. "id == 123", "id > 123", "NAME == 'PAXOS S3'"

"(id > 123) && (NAME == 'PAXOS S3')" (評価順は左側の()から）

isnowait: 排他モード !=0 :NOWAIT =0 : WAIT

(現在、排他は未サポート→ =1で使用）

return値= falseの時エラー

### （６）テーブル削除

**boolean cmdb\_drop(tablename)**

tablename: テーブル名

return値= falseの時エラー

### （７）検索条件設定

**bool cmdb\_search(tablename, condition)**

tablename: テーブル名

condition: 検索条件文字列。

左辺にカラム名

比較演算： ==, !=, <, >, <=, >=

論理演算： &&, ||

条件値の文字列は ' で括る

Ex. "id == 123", "id > 123", "NAME == 'PAXOS S3'"

return値= falseの時エラー

### （８）検索条件データ取得　(cmdb\_serchの後で）

**Array(Hash) cmdb\_find(tablename, maxrec, isshare)**

tablename: テーブル名

maxrec: 取得レコード最大数

isshare: 排他モード !=0 :SHARE =0 : EXCLUSIVE

(現在、排他は未サポート→ =1で使用）

return値= nilの時エラー

Array(Hash): key=カラム名、val=値（type対応のruby Object)

検索データが1件もない場合は、長さゼロの配列が返される。

### （９）SQL文の実行試験 (1-SQL文の実行限定）

**bool cmdb\_sqltest(sqlstatement)**

sqlstatement: sqlコマンド文字列

return値= falseの時エラー

　※実行結果は、無条件に廃棄されて終了する

## テーブルクローズ

**bool cmdb\_close(tablename)**

tablename: テーブル名

return値= falseの時エラー

## CMDBとの切断

**bool cmdb\_unlink()**

return値= falseの時エラー

## 【Rubyライブラリ使用例】

cmdbaccess\_test.rb

require "./CmdbAccess.so"

ttt = CmdbAccess.new

################ Prologue

retd = ttt.cmdb\_prologue("CMDBTEST-001")

print "\*\*\* cmdbaccess\_test.rb: after cmdb\_prologue: return=[",retd,"]\n\n"

### Connect to CMDB

retd = ttt.cmdb\_link("CLIENT01","shio\_cmdb")

print "\*\*\* cmdbaccess\_test.rb: after cmdb\_link: return=[",retd,"]\n\n"

###########################################

### Table DROP before CREATE Run

retd = ttt.cmdb\_drop("cmdb\_test")

print "\*\*\* cmdbaccess\_test.rb: after cmdb\_drop: return=[",retd,"]\n\n"

### TABLE CREATE

retd = ttt.cmdb\_create("cmdb\_test", "name char(32), idata int, name2 char(32)")

print "\*\*\* cmdbaccess\_test.rb: after cmdb\_create: return=[",retd,"]\n\n"

### TABLE OPEN

retd = ttt.cmdb\_open("cmdb\_test")

print "\*\*\* cmdbaccess\_test.rb: after cmdb\_open: return=[",retd,"]\n\n"

###########################################

### INSERT

array = Array.new

hash = {"name" => 'name-A', "idata" => 123, "name2" => 'char data-A'}

array.push(hash)

hash = {"name" => 'name-B', "idata" => 234, "name2" => 'char data-B'}

array.push(hash)

hash = {"name" => 'name-C', "idata" => 345, "name2" => 'char data-C'}

array.push(hash)

print "\*\*\* cmdbaccess\_test.rb: array=[",array,"]\n"

retd = ttt.cmdb\_insert("cmdb\_test", array, 0)

print "\*\*\* cmdbaccess\_test.rb: after cmdb\_insert: return=[",retd,"]\n\n"

### sqlselect

retd = ttt.cmdb\_sqlselect("select \* from cmdb\_test;", 10)

print "\*\*\* cmdbaccess\_test.rb: after cmdb\_sqlselect(cmdb\_test): return=[",retd,"]\n\n"

### sqlselect

retd = ttt.cmdb\_sqlselect("select \* from cmdb\_test where name = 'name-B';", 10)

print "\*\*\* cmdbaccess\_test.rb: after cmdb\_sqlselect(cmdb\_test) WHERE: return=[",retd,"]\n\n"

### DELETE

retd = ttt.cmdb\_delete("cmdb\_test", "idata == 234", 0)

print "\*\*\* cmdbaccess\_test.rb: after cmdb\_delete(cmdb\_test): return=[",retd,"]\n\n"

### sqlselect

retd = ttt.cmdb\_sqlselect("select \* from cmdb\_test;", 10)

print "\*\*\* cmdbaccess\_test.rb: after cmdb\_sqlselect(cmdb\_test): return=[",retd,"]\n\n"

###########################################

### CLOSE

retd = ttt.cmdb\_close("cmdb\_test")

print "\*\*\* cmdbaccess\_test.rb: after cmdb\_close: return=[",retd,"]\n\n"

#### Disconnect from CMDB

retd = ttt.cmdb\_unlink()

print "\*\*\* cmdbaccess\_test.rb: after cmdb\_unlink: return=[",retd,"]\n\n"

######### Epilogue

retd = ttt.cmdb\_epilogue()

print "\*\*\* cmdbaccess\_test.rb: after cmdb\_epilogue: return=[",retd,"]\n\n"

root@paxos034:/home/nami/PaxosS3/RailsApp/LIB/CMDB#

# API（PHPライブラリ）

下記記法で　PHPtblobj : r\_tbl\_t 構造体のオブジェクトを意味する。

## CMDBとの接続

**boolean init(cellname)**

cellname: CELL名 or ipaddr:port (ex. 192.168.222.1:2000)

return値= falseの時エラー

## CMDBとの切断

**boolean final()**

return値= falseの時エラー

## テーブル作成

**PHPtblobj\* create(tablename,schema)**

tablename: テーブル名

schema: レコード定義スキーマ

(ex. "col1 int,col2 long,col3 char(12),col4 ulong")

※定義可能型: int, uint, long, ulong, number(=ulong), float, char,

varchar, date(=long)

return値= falseの時エラー

r\_tbl\_t オブジェクト

## テーブル削除

**boolean drop(tablename)**

tablename: テーブル名

return値= falseの時エラー

## テーブルオープン

**PHPtblobj\* open(tablename)**

tablename: テーブル名

return値= falseの時エラー

r\_tbl\_t オブジェクト

## テーブルクローズ

**boolean close()**

tablename: テーブル名

return値= falseの時エラー

## DBに対するコミット

**boolean commit()**

return値= falseの時エラー

## DBに対するロールバック

**boolean rollback()**

return値= falseの時エラー

## テーブル操作関連

### （１）データ登録

**boolean insert(recdatastr, keystring, isshare)**

recdatastr: ()でレコード定義。

(....),(...), ...... のように複数レコード定義

()内はkeystrの定義順でアイテムデータを文字列で定義

Ex. (123,12.34,'abcdfeg', ....)

keystr: レコードのカラム名定義。 col1,col2,col3, ....

isshare: 排他モード !=0 :SHARE =0 : EXCLUSIVE

return値= falseの時エラー

### （２）データ削除

**boolean delete(arrayabs, isnowait)**

arrayabs: 削除したいレコードインデックス。配列。

isnowait: 排他モード !=0 :NOWAIT =0 : WAIT

(現在、排他は未サポート→ =1で使用）

return値= falseの時エラー

### （３）データ更新

**boolean update(arrayabs, recdatastr, isnowait)**

arrayabs: 更新したいカレントのテーブル上のレコードインデックス。配列。

recdatastr: ()でレコード定義。

(....),(...), ...... のように複数レコード定義

()内はテーブル中のアイテムデータを定義順に文字列で定義。

アイテムの欠落は不可。

isnowait: 排他モード !=0 :NOWAIT =0 : WAIT

(現在、排他は未サポート→ =1で使用）

return値= falseの時エラー

### （４）検索条件の設定

**boolean search(condition)**

condition: 検索条件文字列。

左辺にカラム名

比較演算： ==, !=, <, >, <=, >=

論理演算： &&, ||

条件値の文字列は ' で括る

Ex. "id == 123", "id > 123", "NAME == 'PAXOS S3'"

return値= falseの時エラー

### （５）レコードデータ取得（search()で設定した結果を得る）

**PHPstringObj find(nl, isshare)**

nl: 最大取得レコード数

isshare: 排他モード !=0 :SHARE =0 : EXCLUSIVE

return値= falseの時エラー

データの文字列。

()でレコード定義。

(....),(...), ...... のように複数レコード定義

()内はテーブル中のアイテムデータを定義順に文字列で定義。

アイテムの欠落は不可。

### （６）検索用カーソルを先頭に設定

**boolean rewind()**

return値= falseの時エラー

### （７）テーブルのソート（search()で検索条件にソート条件を追加）

**boolean sort(arraysort)**

arraysort: key=カラム名、val=ソート条件のハッシュレコードの配列。

ソート条件：昇順（R\_ASCENDANT）、降順（R\_DESCENDANT）

return値= falseの時エラー

### （８）テーブルの圧縮

**boolean compress()**

return値= falseの時エラー

### （９）レコード数の参照

**PHPlongObj getrecno()**

return値= falseの時エラー

登録済みレコード数

### （１０）スキーマ定義参照

**PHPstringObj getschema()**

return値= falseの時エラー

レコード定義スキーマ文字列

(ex. "col1 int,col2 long,col3 char(12),col4 ulong")

### （１１）テーブル名参照

**PHPstringObj getname()**

return値= falseの時エラー

テーブル名

### （１２）カレントテーブルに対するコミット

**boolean flush()**

return値= falseの時エラー

### （１３）カレントテーブルに対するロールバック

**boolean cancel()**

return値= falseの時エラー

### （１４）SQL文実行

**boolean SqlExecute(sqlstatement)**

sqlstatement: sqlコマンド文字列

return値= falseの時エラー

### （１５）SQL文実行の先頭の結果取得

**PHPtblobj\* SqlResultFirstOpen()**

return値= falseの時エラー

r\_tbl\_t オブジェクト（結果データテーブル）

### （１６）SQL文実行の次の結果取得

**PHPtblobj\* SqlResultNextOpen()**

return値= falseの時エラー

r\_tbl\_t オブジェクト（結果データテーブル）

### （１７）SQL文実行の結果削除

**boolean SqlResultClose()**

return値= falseの時エラー

## 【PHPライブラリ使用例】

<?php

// New Object

$cmdb = new cmdb();

// Connect to CMDB

$cmdb->init('cmdb');

// Table CREATE

$tp=$cmdb->create('test\_tbl', 'u\_id int,u\_name varchar(256), u\_pass char(32)');

// Table OPEN

$tp=$cmdb->open('test\_tbl');

// INSERT

$keys= 'u\_id,u\_name';

$keys2= 'u\_id,u\_name';

print("keys -> $keys\n");

print("keys2 -> $keys2\n");

$ret = $tp->insert("(100,'kubo','ma3nari'),(-1,'ohyama','ky0-5')",$keys,false);

$keys= 'u\_id,u\_name';

print("keys -> $keys\n");

$keys2= 'u\_id,u\_name';

print("keys2 -> $keys2\n");

$ret = $tp->insert("(100,'ohyama','ma3nari'),(-1,'kubo','ky0-5')",

$keys2,false);

print("keys -> $keys\n");

// SEARCH

$tp->search(null);

// REWIND

$tp->rewind();

// FIND

$recs=$tp->find(5,false);

if ($recs !== false && is\_array($recs)) {

foreach ( $recs as $rec ) {

printf("record:[".$rec."]\n");

}

}

// GET Record Number

$n = $tp->getrecno();

print( "recode no.[$n]\n" );

// Schema Info. GET

$schema = $tp->getschema();

if ( $schema === false ) {

print("getschema is fail.\n");

} else {

print("schema -> $schema\n");

}

// DELETE

$nrec=$tp->getrecno();

$abs = array();

for ( $i = 0; $i < $nrec; $i++ ) {

array\_push($abs,$i);

}

$tp->delete($abs,false);

// Table CLOSE

$tp->close();

// Table DROP

$cmdb->drop('test\_tbl');

} else {

$tp->flush($tp);

$tp->close($tp);

}

// CMDB DisConnect

$cmdb->final();

?>

# SQL文法

## 型

次の型をサポートする。

1. int 整数型(４バイト)
2. uint 正の整数型(４バイト)
3. long 整数型(8バイト)
4. ulong 正の整数型(8バイト)
5. number(n,m) 整数型(8バイト、n及びmは無視、即ち10進数で19桁となる)
6. float 浮動少数点(4バイト)
7. char(n) 文字列(nバイト、固定長)
8. varchar(n) 文字列(nバイト、可変長最大)
9. date Java型時刻(8バイト、driverで変換処理を行う)

## 文

文１；文２；…

文は、“；”で区切られている。

## schema文

1. create　table

create　table　テーブル名(カラム名　型、…);

primary　keyの指定はできるが、無視される。

カラム名は大小文字が区別されるが、型は大小文字が区別されない。

なお、すでにテーブルが存在しオープンされていない場合には、カラム名の一致するものについて自動的にデータ移行を行う。

1. create　view

create　view　ﾋﾞｭｰ名　as　select　表示ﾘｽﾄ　from　ﾃｰﾌﾞﾙﾘｽﾄ　[where　条件式]　[union　all　select　…];

ビューはクエリとして定義され、更新は不可である。

Select以下は、select文を参照されたい。

なお、更にOrder　by句も指定できる。

また、ビューについてはテーブルリストにあるテーブルが更新されない限り、再構築されない。

Union以下は、分割ビューであり、複数テーブルを和結合する。ただし、スキーマは同じでなければならない。

なお、すでにビューが存在しオープンされていない場合にはビューを作り変える。

1. drop　table

drop　table　テーブル名;

テーブルを削除する。

1. drop　view

drop　view　ビュー名;

ビューを削除する。

## select文

select　表示ﾘｽﾄ　from　ﾃｰﾌﾞﾙﾘｽﾄ　[where　条件式]　[order　by　ｵｰﾀﾞｰﾘｽﾄ];

表示リストは、“\*”とすると、テーブルリストの全てのカラム名が表示される。表示リストは、”テーブル名.カラム名　表示名“で、”、“のカンマリストで定義する。表示名は省略可能であり、また、”テーブル名.”を省略すると最初のテーブルと解釈される。エイリアス名が指定されていればテーブル名ではなくエイリアス名を使用する。

テーブルリストは、参照するテーブルをカンマリストで定義する。なお、エイリアスも指定できる。

エイリアスは、

ﾃｰﾌﾞﾙ名　ｴｲｱﾘｱｽ名、…

で指定する。

where句は、省略可能である。条件は、複数テーブルがある場合には、”テーブル名.カラム名”とする。条件式には、結果が論理演算となる論理式、比較式が許される。演算子としては、（、）、＝、！＝、＜＝、＞＝、AND,ORである。結合は全ての組み合わせについて調べられ、真となる組み合わせだけが取り出される。UNIONの機能はない。

文字列に関して、like文を指定できる。

ｶﾗﾑ名　like　ﾊﾟﾀｰﾝ文字列

ｶﾗﾑ名　like　ﾊﾟﾀｰﾝ文字列　escape　文字

ｶﾗﾑ名　not　like　ﾊﾟﾀｰﾝ文字列

ｶﾗﾑ名　not　like　ﾊﾟﾀｰﾝ文字列　escape　文字

パターン文字列、文字は、’’で囲む。

パターン文字列には、％、＿を指定することができ、カラム名で指定される文字列にパターン文字列が含まれるか否かを調べる。

order　by句も省略可能である。”テーブル名.カラム名　asc|desc”のカンマリストで指定する。ascは昇順、descは降順である。order　byに指定がない場合は検索された順番となる。ascあるいはdescの指定がない場合には、asc(昇順)となる。

”テーブル名.”は、省略できるが最初のテーブル名とされる。

## insert文

insert　into　ﾃｰﾌﾞﾙ　values(ﾊﾞﾘｭｰﾘｽﾄ);

insert　into　ﾃｰﾌﾞﾙ　(ｶﾗﾑﾘｽﾄ)　values(ﾊﾞﾘｭｰﾘｽﾄ);

バリューリストは値のカンマリストであり、カラムリストはカラム名のカンマリストである。

カラムリストがない場合は、カラムの定義順に値が設定される。

なお、文字列は、「‘’」で囲む。文字列中に「‘」を指定したい場合には「‘’」とする。

## update文

update　ﾃｰﾌﾞﾙ　set　ｾｯﾄﾘｽﾄ　[where　条件];

where条件の成立するレコードに対して、セットリストのカラムが更新される。Where条件がない場合には全レコードが対象となる。

セットリストは、

カラム名＝算術式

のカンマリストである。

算術式には、カラム名を変数とすることができる。例えば、

A=A+B+1

とすることができる。

最初の更新レコード(旧値)が返却され、更新はatomicである。したがって、シーケンサーに使用できる(例、version=version+1)。

## delete文

delete　from　ﾃｰﾌﾞﾙ　[where　条件];

where条件の成立するレコードが削除される。

Where条件の指定がない場合は、全レコードが対象となる。

## transaction文

begin;

commit;

rollback;

beginはトランザクションの開始を宣言するが、commit/rollbackで暗黙に開始は宣言される。

Commitは更新レコードを確定し、rollbackは更新レコードをトランザクション開始前にもどす。

トランザクション管理は接続単位で行われる。

複数が参照する場合には、更新中のデータが参照される。これを避けるためには、別途用意されている排他制御機構を用いる。

# 試験

　試験用シェルスクリプトは、$bin/cmdに配備されている。Bの文字のないシェルスクリプトは機能試験用であり、Bの文字があるシェルスクリプトは性能試験用である。

　単体の場合は、$bin/cmdで起動する。$bin/cmd配下のコマンドが実行される。

　セルの場合は、$bin/paxosでTestPaxos.shで起動する。TestPaxos.shは、$bin/cmdのシェルスクリプトを呼び出す。このシェルスクリプトは$bin/paxos配下のコマンドを実行する。

例

TestPaxos.sh　セル名　Test1

セル名のセルに対して、「Test1.sh　セル名」を実行する。

# 付録1　NWGadget.TAR、xjPing.TARのbuild方法

1. 開発環境の整備

makeコンパイル環境、bison及びlibreadline-devを整備する。

1. ディレクトリの作開発環境の整備

PAXOS/NWGadget、PAXOS/Paxos\_1を作成しておく。

1. NWGadget.TAR

PAXOS/NWGadgetにコピーし、「make」とする。ライブラリが作成される。

1. xjPing.TAR

PAXOS/Paxos\_1にコピーし、「make　–f　Makefile.xjPing」とする。

1. 構成ファイルの作成

PAXOS/Paxos\_1/xjPing/bin/paxosに移動し、サンプルの構成ファイルである\_xjPingPaxos.confを参照し、新しい構成定義を行う。

1. 配備

deploy-xjping-bin.sh、deploy-xjping-conf.shでロードモジュールと構成ファイルを配備するが、配備先のディレクトリが作成できるようにパーミッション等に注意し整備する。配布版は、3台コア1台拡張の4台構成であり、最低2台で動作する。

1. サーバの起動

ssh-start.shで順次起動する。