ｘｊ Ping@IRIS 使用手引書

2003/07/09
2003/08/10
2003/11/26
2004/07/26（改版、旧版と互換性なし）
2004/08/05
2004/09/10

版	修正
2003/11/26	データベースサーバ名の環境変数は大文字で指定する。
2003/12/8	リテラル中の‘ ’は‘と変換する。
2003/12/8	xjSql に終了コマンド「quit;」を追加する。
2003/12/8	xjShutdown コマンドを追加する。
2004/5/12	xjPing の修正(float の>=評価式が抜けていた)。
2004/7/8	Where 句で存在しないカラムを指定した時のエラー制御を修正する。
2004/7/26	Csv ファイルの入出力のコマンド xjExport/xjImport を追加する。DB のスキーマを変更したため、旧版と互換性はない。
2004/7/26	Like 文を追加する。
2004/8/5	xjSql の入力で BS(バックスペース)を有効にする。
2004/9/10	Where 文での a.seq=b.e1　or　a.seq=b.e2 のようなﾃｰﾌﾞﾙ対の重複時の障害を修正する。(xj_hash.c)

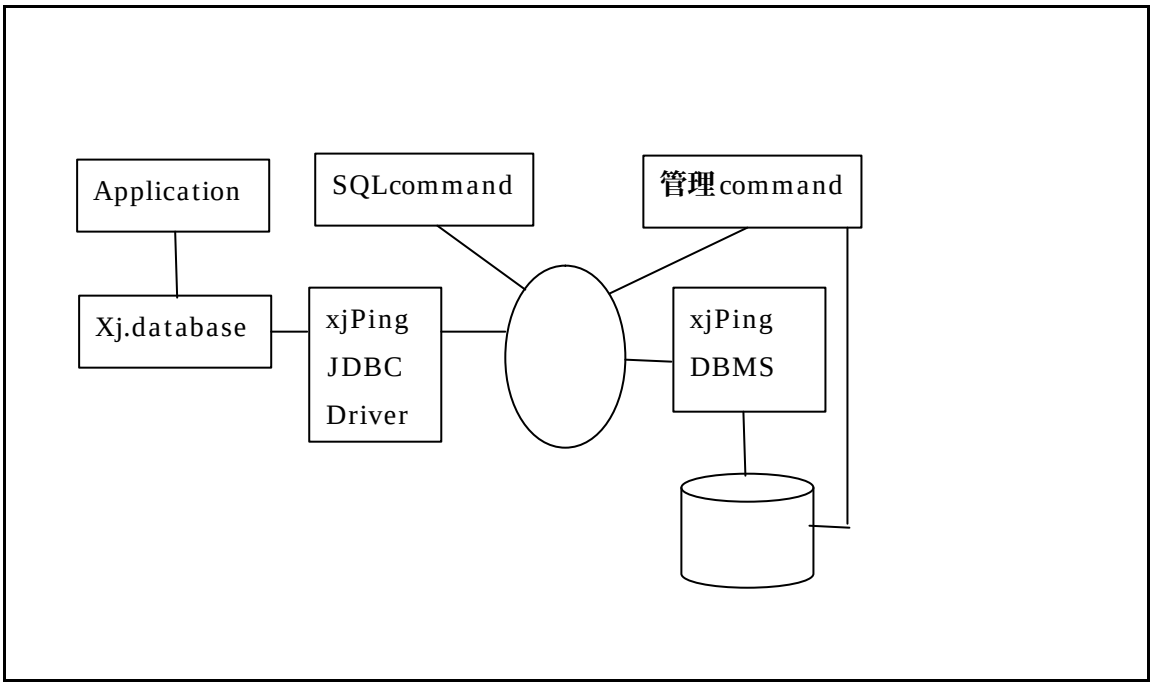
1. 概要

xjPing は、Reference Implementation 用に超高速超軽量を目的として開発された RDBMS であり、主に小規模中規模向けであるが、swap 機能も有し大規模システムにも適用できるように設計されている。

xjPing は、サーバ及び管理コマンド、SQL コマンド、JDBC ドライバ、クライアント C ライブラリ[1]より構成される。

JDBC ドライバは、xj.database パッケージを支援する。

1.1. 構成



2. DBMS サーバ(xjPing)

2.1. 起動

起動は、

xjPing　データベースサーバ名　パス名　[最大メモリ量]

で行う。

ここで、データベースサーバ名については環境変数で大文字で次のように与える。

データベースサーバ名＝IP アドレスあるいはホスト名[：ポート番号]

“：ポート番号”を指定しない場合は、4989 となる。

例えば、

[1] C ライブラリについては、現在は非公開です。

dbServer1=10.1.1.1：4989

とする。

xjPing 及び管理コマンドは、getenv(データベースサーバ名)で情報を取得する。

JDBC ドライバでは、次のように指定する。

jdbc：xjsql：//IP アドレス：ポート番号/xjDB

パス名は、データを格納するディレクトリであり、テーブル名はファイル名に対応する簡明な表現となっている。

最大メモリ量は、サーバがキャッシュに消費する目安を指定する。デフォルトは、最大１GB を前提としているが、必要に応じて１MB より順次拡張される機構となっている。

なお、オープンされているテーブルはキャッシュされるが、最大メモリ容量を超えた場合には自動的にスワップアウト/スワップインされる。

2.2. 停止と再開始

停止は、

xjStop　データベースサーバ名

で行い、再開始は、

xjRestart　データベースサーバ名

で行う。

終了は

xjShutdown　データベースサーバ名

あるいは、xjStop 正常終了後、プロセスキルによる。

3. SQL

3.1. SQL コマンド

SQL コマンドは、

xjSql　データベースサーバ名

で実行する。

Sql コマンドの投入は、標準入力より行われる。

終了は「quit;」あるいは「＾D」とする。

3.2. SQL 文法

3.2.1. 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)
⑧ varchar(n)　　文字列(n バイト)
⑨ date　　Java 型時刻(8 バイト、driver で変換処理を行う)

3.2.2. 文

文１；文２；...

文は、“；”で区切られている。

Driver 使用時は、driver が自動付加する。

3.2.3. 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、開発の便宜のため、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。

② 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、開発の便宜のため、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。

③ drop table

drop　table　テーブル名;

テーブルを削除する。

④ drop　view

drop　view　ビュー名;

ビューを削除する。

3.2.4. 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。

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算術式、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。

3.2.5. insert 文

insert into テーブル values(バリューリスト);

insert into テーブル (カラムリスト) values(バリューリスト);

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。

3.2.6. update 文

update テーブル 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。

本版では、最初の更新レコードが返却される。これにより、シーケンサー等の atomic 処理が可能となる。

3.2.7. delete 文

delete from テーブル [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。

3.2.8. 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 DBMS 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。

4. 排他制御

本 DBMS は、複数テーブルに対する同時排他制御機構、テーブルの複数レコードに対する同時排他制御機構を有しているが、jdbc ドライバでは資源に対する排他制御機構を提供する。

このため、資源管理のための“XJ_LOCK”システムテーブルが前提とされ、初期構築をしておく必要がある。

例えば、初期化 SQL 文で次のようにする。

```
Drop table XJ_LOCK;
Create table XJ_LOCK( NAME char(16));
Insert into XJ_LOCK values('LockResource1');
Insert into XJ_LOCK values('LockResource2');
```

ドライバでは、XjLock クラスのクラスメソッドとして実装されている。

4.1. Connection の取得

コネクションを

```
Class.forName("xj.sql.XjDriver");
Connection DriverManager.getConnection("jdbc:xjsql://192.168.0.4:4989/xjDB");
```

で取得する。

4.2. 資源の生成と削除

XjLock.create(Connection,資源名)

で資源を生成し、

XjLock.delete(Connection,資源名)

で削除する。

4.3. 排他の宣言

XjLock.lock(Connection,資源名)

XjLock.lockWithoutWait(Connection,資源名)

で排他を宣言する。

lock は解除まで待つが、lockWithoutWait は解除を待たず、排他宣言されている場合には XjLockBusyException が発生する。

4.4. 排他の解除

XjLock.unlock(Connection,資源名)

で排他を解除する。

5. 管理コマンド

5.1. xjInfo

xjInfo データベースサーバ名

サーバ状態、メモリ使用状況、ポート及びユーザ状況、オープンテーブル情報を取得する。

5.2. xjTable

xjTable データベースサーバ名 テーブル名 [開始レコード 終了レコード]

サーバ上のテーブル情報を取得する。

開始レコード、終了レコードを省略すると、全レコードの情報が表示される。

開始レコード＞終了レコードとするとレコードは表示されない。

5.3. xjFile

xjFile テーブルパス名 [h]

サーバを介さず、テーブルファイルから直接情報を取得する。

h(head)を指定するとレコードは表示されない。

5.4. xjCompress

xjCommpress テーブルパス名

テーブルファイルを圧縮する。

5.5. xjList

xjList データベースサーバ名

データベースのテーブル名一覧を取得する。

5.6. xjExport

xjExport データベースサーバ名 テーブル名 [レコード数]

csv ファイルをテーブル名.csv として出力する。レコード数は 1 回の要求で取得する最大レコード数であり、指定がなければ 100 としている。

なお、スキーマの出力はない。

5.7. xjImport

xjImport データベースサーバ名 テーブル名 csv ファイル名 [レコード数]

csv ファイルをﾃｰﾌﾞﾙに取り込む。レコード数は 1 回の要求でテーブルに追加する最大レコード数であり、指定がなければ 100 としている。

データは、テーブルのスキーマに従って変換される。

6. 規模と性能

6.1. テーブル

１テーブルは、１ファイルとして保存されるので最大１G バイト程度のサイズを保存できる。

6.2. メモリ

本サーバでは、実テーブルに対しては全データ、検索で生成される一時テーブルに対してはセグメント方式を採用しているが、オンメモリで処理することを前提としている。したがって、メモリ容量は少なくとも同時に使用するテーブルのサイズ以上を目安とする必要がある。

更新中のテーブルでは、メモリが足りない場合には swap out され、必要なときに swap in される。Swap file は XJ_SWAPn である[2]。

6.3. 性能

800MHz の CPU、384MB のメモリで、２テーブルの join8,000 万件の検索で 3.5 分、2.6μs である。

7. 性能向上のヒント

7.1. 分割ビューと UNION

高速化を図るためオンメモリ処理を行っている。

特に、複数テーブルの join については分割統治を前提としており、インデックス[3]を採用していないため、総当り方式となっている[4]。分割統治を行わないと無駄な処理を行うこととなる。

これを解決するためには、ビューを利用した分割統治方式をアプリケーションで採用することが勧められる。

例えば、

```
select * from T_1, T_2 where T_1.INT = T_2.INT;
```

[2] 現在は、スワップファイルは 1 つである。

[3] 10 万件程度まではインデックス処理はコストが高い。

[4] 但し、「=」については、ハッシュによる高速化を行っている。また、依存テーブルに変更がない場合にはビューは一時ファイルとしているので、この時はテーブルと同じ扱いとなる。

は、

```
create view T_10 as select * from T_1 where T_1.INT < 1000;
create view T_20 as select * from T_2 where T_2.INT < 1000;
create view T_11 as select * from T_1 where T_1.INT >= 1000;
create view T_21 as select * from T_2 where T_2.INT >= 1000;
```

とし、さらに

```
create view T0 as
        select * from T_10, T_20 where T_10.INT=T_20.INT;
create view T1 as
  select * from T_11, T_21 where T_11.INT=T_21.INT;
```

を分割ビューを定義し、そして、union で統合ビューを定義する。

```
  create view T as
      select * from T0
      UNION ALL
      select * from T1
      UNION ALL
      select * from T2
      ○○○
      ;
```

とする。

アプリケーションでは、この UNION 結合の統合ビューを参照するだけでよい。