ホワイトペーパー

Microsoft SQL Server から SQL Anywhere® インメモリ・データベースへのデータのダウンロード

Breck Carter

目次

このホワイトペーパーは、他のさまざまなデータベースから SQL Anywhere バージョン 11 インメモリ・データベースにデータをダウンロードし、インメモリ・プロセスのシャットダウン前に元のデータベースにデータを保存またはアップロードするさまざまな手法について紹介する連載記事の第 1 回です。

具体的には、Microsoft SQL Server 2008 データベースから、「チェックポイントのみ」モードとは対照的な「書き込み禁止」モードのインメモリ操作を実行する SQL Anywhere データベースへのデータのダウンロードについて説明します。5 つの手法を紹介し、それぞれについて、重要な一連のデータに対してテストを行った実際のコードを示します。データとして、2 つの異なるデータベース製品間のインタフェースにおいて予測外の状況が発生する可能性があるものを選択しました。

これは根本的にハウツー記事です。「何が最善であるか」という結論は導き出していません。これは、その答えが読者それぞれの優先順位によって決まるからです。実際、同じアプリケーションでも、1 つは高速化のため、もう 1 つは柔軟性のためなど、目的に応じて複数の手法を使用することもあります。

使用手法の概要

- 手法 1 は 2 つの手順からなるプロセスです。最初に、高パフォーマンス SQL Server の BCP ユーティリティ (Bulk Copy Program ：一括コピー・プログラム) を使用して、SQL Server ソース・テーブルを LAN 経由で ASCII テキスト・ファイルにアンロードします。

  次に、SQL Anywhere の高パフォーマンスの LOAD TABLE 文を使用して、そのファイルからインメモリ・データベース内のターゲット・テーブルにデータをロードします。

  BCP の “queryout” オプションを使用して、単純なクエリ “SELECT * FROM main.dbo.mss_source” を実行します。データ自体にタブ、カンマ、引用符、改行などの特殊文字が含まれる状況に対処するために、テキスト・ファイル内では、ローおよびカラムの区切り文字列として特殊文字の文字列を使用します。

```
“c:¥Program Files¥Microsoft SQL
  Server¥100¥Tools¥Binn¥bcp.exe”^ “SELECT * FROM
  main.dbo.mss_source”^
  queryout “¥¥PAVILION2¥C¥data¥main¥mss_source.txt”^
  -c^
  -t $#$¥t^
  -r $#$¥n^
  -P j68Fje9#fyu489^
  -S BRECK-PC¥TSUNAMI^
  -U sa
1925469 rows
copied. LOAD
TABLE
sa_target
  FROM
  ‘c:/data/main/mss_source.txt’
  DELIMITED BY ‘$#$¥x09’
  ESCAP
  ES OFF
  QUOTE
  S OFF
  ROW DELIMITED BY ‘$#$¥x0D¥x0A’;
```

- 手法 2 では、Mobile Link を使用して、SQL Server から SQL Anywhere にダウンロードのみの同期を実行します。Mobile Link™ 同期プロセスは、クライアント・コンピュータで Mobile Link クライアント dbmlsync.exe が実行されると起動され、クライアント・コンピュータからサーバ・コンピュータ上で実行中の Mobile Link サーバへの TCP/IP 接続が確立されます。さらに、Mobile Link サーバは、SQL Server データベースに ODBC 接続し、事前に定義された “download_cursor” (次のコードを参照) という SQL スクリプトを実行して mss_source テーブルのすべてのローを選択します。これらのローは Mobile Link クライアントに送信され、そこで SQL Anywhere データベースに挿入されます。

```
USE main
GO

EXECUTE ml_add_table_script
  ‘v1’,
  ‘sa_target’,
  ‘download_cursor’,
  ‘SELECT * FROM mss_source’
```

- 手法 3 では、「リモート・データ・アクセス」とも呼ばれる SQL Anywhere プロキシ・テーブルを使用して、SQL Server 上の mss_source テーブルの全ローを SQL Anywhere 上の sa_target テーブルにコピーします。

  これは「プル」プロセスです。このプロセスにより、SQL Server 上のテーブルを指すプロキシ・テーブルが SQL Anywhere 上に定義され、INSERT ... SELECT 文が SQL Anywhere で実行されます。

```
CREATE EXISTING TABLE proxy_mss_source
   AT 'mss.main.dbo.mss_source';

INSERT sa_target
SELECT *
   FROM proxy_mss_source;
```

- 手法 4 では、SQL Server のリンク・サーバ機能を使用して、すべてのローを SQL Server から SQL Anywhere へ「プッシュ」します。これは手法 3 と似ていますが、SQL Server 上に「プロキシ・テーブル」が定義されず、代わりに SQL Server 上で実行される INSERT ... SELECT により SQL Anywhere テーブルが明示的に指定されます。

```
EXEC sp_addlinkedserver
   @server = 'mem',
   @srvproduct = 'xxx',
   @provider = 'MSDASQL',
   @datasrc = 'sa_system_dsn'
GO

INSERT INTO mem..dba.sa_target
SELECT *
   FROM mss_source
GO
```

- 手法 5 では、SQL Server の OPENROWSET 構文を使用して手法 4 と同じ操作を実行します。ただし、リンク・サーバの定義は不要です。コードの外見はまったく異なりますが、実際に使用される技術は同じです。

```
INSERT INTO OPENROWSET ( 'MSDASQL',
   'sa_system_dsn'; 'dba';
   'sql', dba.sa_target )

SELECT *
   FROM mss_source
GO
```

- 手法 4 と手法 5 については 2 回取り上げますが、2 回目の説明では、OLE DB プロバイダとして別の SAOLEDB.11 プロバイダを使用します。このプロバイダは SQL Anywhere に同梱されており、SQL Server に付属の MSDASQL プロバイダに代わる、より高速なプロバイダです。

テストのセットアップ

この項では、テストのセットアップについて詳しく説明します。「手法 1」の項まで直接進み、必要に応じてこの項に戻ってセットアップについて不明な点を確認することもできます。

これは、「ベンチマーク」の定義をどのように拡大解釈しても、ベンチマーク・パフォーマンス・テストのセットアップではありません。また、SQL Anywhere と SQL Server を比較するものでもありません。ここで説明するすべての手法では両製品を扱っていますが、パフォーマンスを示す図は、2 つの製品のうちどちらを選択すべきかではなく、さまざまな手法からどの手法を採用するかを決定する際に役立てていただくことを目的としています。さらに、パフォーマンスが唯一の決定要因となることはほとんどなく、パフォーマンスよりも柔軟性や単純さなど他の基準に基づくことで、より適切な手法が判明することもあります。

テスト環境の構築に使用したハードウェアとソフトウェアは次のとおりです。

サーバ・コンピュータ
- Whitebox デスクトップ (Intel Core 2 Quad Q9450 2.66Ghz 4G RAM 搭載)
- Windows Vista Ultimate 64 ビット (ビルド 6001 SP1)
- SQL Server 2008 Enterprise Edition 64 サービス・パック 1
- ホスト名 “BRECK-PC”
- SQL Server 名 “TSUNAMI”
- データベース名 “main”
- SQL Anywhere 11.0.1.2276 の SAOLEDB.11 プロバイダ

クライアント・コンピュータ
- HP Pavilion ラップトップ (4GHz Pentium 4、2G RAM 搭載)
- Windows XP SP2
- SQL Anywhere 11.0.1.2276
- ホスト名 “PAVILION2”
- SQL Anywhere サーバ名 “mem”
- データベース名 “mem”

Windows、SQL Server 2008、および SQL Anywhere 11 のインストールには、SQL Server の起動に必要なすべてのサービスも含め、標準セットアップを使用しました。ここでは、これらの標準セットアップについては説明しませんが、SQL Anywhere サーバを起動するコマンドなど、その他すべてについて示しています。

多くの部分では、SQL 文および Windows コマンドのテキスト・スクリプトを、それと同等のタスクを実行する GUI ダイアログと対比させて示しています。テキスト･スクリプトを使用した理由は 2 つあります。第 1 に、スクリプトはより簡潔なため説明しやすく、スクリプト構文の変更頻度が GUI レイアウトと比べて少ない傾向にあることです。第 2 に、スクリプトは、プロセスの実装が繰り返し必要な多くの DBA が使用しています。GUI は、新たな対象の学習や一度限りのタスクの実行には最適ですが、繰り返しタスクの自動化については、スクリプトの方が大幅に優れていることも珍しくありません。また、スクリプトではコメントの記述が可能で、それ自体が制御変更に役立ちます。

念のため記しておきますが、同等の GUI ダイアログは、インストールした次のプログラムに含まれています。
- [スタート]- [すべてのプログラム] - [Microsoft SQL Server 2008] - [SQL Server Management Studio]
- [スタート] - [すべてのプログラム] - [Microsoft SQL Server 2008] - [Configuration Tools] – [SQL Server Configuration Manager]
- [スタート] - [すべてのプログラム] - [SQL Anywhere 11] – [Sybase Central]

図 1 に、この記事で多くの SQL スクリプトを実行するために使用した SQL Server コマンドライン SQL ユーティリティの起動方法を示します。

図 1 : SQL Server の SQL コマンド・ユーティリティの起動 | コンテキスト : SQL Server

```
1  "c:¥Program Files¥Microsoft SQL Server¥100¥Tools¥Binn¥sqlcmd.exe"^
2     -d   main^
3     -I^
4     -P   j68Fje9#fyu489^
5     -S   BRECK-PC¥TSUNAMI^
6     -U   sa
```

図 1 の 1 行目は、sqlcmd.exe のファイル仕様です。Microsoft によると、これは旧式の osql.exe に代わるものですが、その動作は非常によく似ています。キャレット “^” は Windows のコマンドライン継続文字で、この記事では、スクリプトが見やすく、また記述しやすいため使用しています。

2 行目では、最初に SQL Server に接続するときに使用するデータベースを指定しています。SQL Anywhere とは異なり、SQL Server では、サーバに接続してから使用データベースを指定し、その後、接続を維持したままデータベースを切り替えることができます。一方、SQL Anywhere では、接続をサーバ内の特定のデータベースに対して確立し、別のデータベースを使用する場合は、同じサーバ上のデータベースであっても新たに接続を開始する必要があります。

3 行目では、SQL コマンド内の識別子を「二重引用符」で囲むことができます。識別子のいずれかが予約語である場合は、この作業が必要になります。

4、5、および 6 行目では、sqlcmd からSQL Server への接続に使用するパスワード、サーバ、およびユーザ ID を指定しています。BRECK-PC はサーバ・コンピュータのホスト名、TSUNAMI は SQL Server データベース・サーバ名です。

図 2 は、ソース・データベースを作成するために使用した SQL Server CREATE DATABASE 文です。データおよ

```
1  CREATE DATABASE main
2  ON PRIMARY
3  ( NAME = main_dat,
4    FILENAME = 'E:¥data¥main¥main.mdf',
5    SIZE = 2GB,
6    FILEGROWTH = 200MB )
7  LOG ON
8  ( NAME = 'main_log',
9    FILENAME = 'E:¥data¥main¥main.ldf',
10   SIZE = 2GB,
11   FILEGROWTH = 200MB )
12 GO
```

図 2 の 1 行目では、データベース名 “main” を指定しています。2 ～ 6 行目ではデータ・ファイルの場所、7 ～ 11 行目ではログ・ファイルの保存先を指定しています。この記事の残りの部分では、データベース名 “main” のみが重要になりますが、それ以外の場合は必ずしもそうとは限りません。

サーバ・コンピュータ上の余分なディスク I/O を避けるため、[コントロール パネル] - [システム] - [システムの保護] – [自動復元ポイント] で Windows Vista の E: ドライブへの復元ポイントをオフにしました。また、同じ理由から、[コントロール パネル] - [管理ツール] - [サービス] で SQL Server VSS Writer を無効にしました。

一部の手法では、SQL Server の構成をいくつか変更する必要がありました。これらの変更のコードは、後で示します。

図 3 は、SQL Server 上のソース・テーブルの CREATE TABLE です。このテーブルは 63 カラム、1,925,469 ローで構成され、データベースの自動監視プロセスで収集された統計データは約 1 ギガバイトでした。このデータの実際の性質は、データが人工的なテスト・データ・ジェネレータではなく実世界のアプリケーションによって生成されたことを踏まえれば、それほど重要ではありません。ただし、これがこのデータのベンチマーク目的への適合性を主張するものではないことはお断りしておきます。

ヒント： SQL Server と SQL Anywhere との間でデータをやり取りするときは、blob を避けてください。これは、SQL Server 側では TEXT や VARCHAR (MAX) などのデータ型、また SQL Anywhere 側では LONG VARCHAR を使用しないということです。実際、SQL Anywhere では、VARCHAR (8001) またはこれより長いデータとして定義された文字列は避ける必要があります。これらのデータは SQL Anywhere では blob ではありませんが、8000 を制限とする SQL Server では blob として処理されます。これは、SQL Server 側でのパフォーマンスの問題だけでなく、致命的なメモリ・リークを防ぐための提案です。この記事で使用している元のテーブルは、TEXT として定義された 9 カラムからなりますが、実際のデータに 144 文字より長い文字列値は含まれていなかったため、そのすべてを容易に VARCHAR (1000) に変更できました。これは、問題がデータ自体ではなく、データ型に関連していることを示しています。

```
1  CREATE TABLE mss_source (
2    sampling_id BIGINT NOT NULL,
3    sample_set_number BIGINT NOT NULL,
4    connection_number BIGINT NOT NULL,
5    blocker_owner_table_name VARCHAR ( 257 ) NULL,
6    blocker_lock_type VARCHAR ( 32 ) NULL,
7    blocker_owner_name VARCHAR ( 128 ) NULL,
8    blocker_table_name VARCHAR ( 128 ) NULL,
9    blocker_reason VARCHAR ( 1000) NULL,
10   blocker_row_identifier VARCHAR ( 32 ) NULL,
11   current_engine_version VARCHAR ( 1000) NOT NULL,
12   page_size INTEGER NOT NULL,
13   ApproximateCPUTime DECIMAL ( 30, 6 ) NULL,
14   BlockedOn BIGINT NULL,
15   BytesReceived BIGINT NULL,
16   BytesSent BIGINT NULL,
17   CacheHits BIGINT NULL,
18   CacheRead BIGINT NULL,
19   “Commit” BIGINT NULL,
20   DiskRead BIGINT NULL,
21   DiskWrite BIGINT NULL,
22   FullCompare BIGINT NULL,
23   IndAdd BIGINT NULL,
24   IndLookup BIGINT NULL,
25   Isolation_level BIGINT NULL,
26   LastReqTime VARCHAR ( 1000) NOT NULL
27   DEFAULT ‘1900-01-01’,
```

```
28  LastStatement VARCHAR ( 1000) NULL,
29  LockCount BIGINT NULL,
30  LockName BIGINT NULL,
31  LockTableOID BIGINT NULL,
32  LoginTime VARCHAR ( 1000) NOT NULL
33  DEFAULT '1900-01-01',
34  LogWrite BIGINT NULL,
35  Name VARCHAR ( 128 ) NULL,
36  NodeAddress VARCHAR ( 1000) NULL,
37  Prepares BIGINT NULL,
38  PrepStmt BIGINT NULL,
39  QueryLowMemoryStrategy BIGINT NULL,
40  QueryOptimized BIGINT NULL,
41  QueryReused BIGINT NULL,
42  ReqCountActive BIGINT NULL,
43  ReqCountBlockContention BIGINT NULL,
44  ReqCountBlockIO BIGINT NULL,
45  ReqCountBlockLock BIGINT NULL,
46  ReqCountUnscheduled BIGINT NULL,
47  ReqStatus VARCHAR ( 1000) NULL,
48  ReqTimeActive DECIMAL ( 30, 6 ) NULL,
49  ReqTimeBlockContention DECIMAL ( 30, 6 ) NULL,
50  ReqTimeBlockIO DECIMAL ( 30, 6 ) NULL,
51  ReqTimeBlockLock DECIMAL ( 30, 6 ) NULL,
52  ReqTimeUnscheduled DECIMAL ( 30, 6 ) NULL,
53  ReqType VARCHAR ( 1000) NULL,
54  RequestsReceived BIGINT NULL,
55  Rlbk BIGINT NULL,
56  RollbackLogPages BIGINT NULL,
57  TempFilePages BIGINT NULL,
58  TransactionStartTime VARCHAR ( 1000) NOT NULL
59  DEFAULT '1900-01-01',
60  UncommitOp BIGINT NULL,
61  Userid VARCHAR ( 128 ) NULL,
62  previous_ApproximateCPUTime DECIMAL ( 30, 6 ) NOT NULL DEFAULT 0.0,
63  interval_ApproximateCPUTime AS ( COALESCE ( "ApproximateCPUTime", 0 )
64  - previous_ApproximateCPUTime ),
65  previous_Commit BIGINT NOT NULL DEFAULT 0,
66  interval_Commit AS ( COALESCE ( "Commit", 0 )
67  - previous_Commit ),
68  previous_Rlbk BIGINT NOT NULL DEFAULT 0,
69  interval_Rlbk AS ( COALESCE ( Rlbk,0 )
70  - previous_Rlbk ),
71  PRIMARY KEY ( sample_set_number, connection_number ) );
72  GO
```

図 4 は、SQL Anywhere データベース・ファイルの作成、データベースの開始、および Interactive SQL ユーティリティの起動に使用した 3 つの Windows XP コマンドです。

図 4 : SQL Anywhere ターゲット・データベースの作成 | コンテキスト : SQL Anywhere

```
1  "%SQLANY11%¥bin32¥dbinit.exe"^
2    mem.db
3
4  "%SQLANY11%¥bin32¥dbspawn.exe"   -f^
5    "%SQLANY11%¥bin32¥dbsrv11.exe"^
6    -im   nw^
7    -c   1200M^
8    -o   dbsrv11_log.txt^
9    mem.db
10
11 "%SQLANY11%¥bin32¥dbisql.com"^
12   -c   "ENG=mem;DBN=mem;UID=dba;PWD=sql;CON=mem-1"
```

図 4 の 1 行目は、Windows での SQL Anywhere の標準セットアップによりインストールされるデータベース初期化ユーティリティ dbinit.exe の完全なファイル仕様です。ワークステーション・コンピュータには複数バージョンの SQL Anywhere が存在することが多いため、コマンド・ファイルを使用する際は、システム PATH に頼らず、常に完全なファイル仕様を指定することをおすすめします。

ヒント : 環境変数 SQLANY11 を使用することで、SQL Anywhere 実行プログラムを実行する Windows コマンド・ファイル内でのファイル仕様のコーディングが容易になります。この環境変数は、標準 SQL Anywhere 11 セットアップにより次のように作成されます。

```
SQLANY11=C:¥Program   Files¥SQL   Anywhere   11
```

2 行目では、SQL Anywhere データベース・ファイルのファイル名を指定しています。トランザクション・ログ・ファイル名はデフォルトで mem.log になり、ページ・サイズはデフォルトで、多くのアプリケーションに最適な 4K になります。

4 行目はオプションです。"spawn" プログラム dbspawn.exe を起動し、さらにこのプログラムにより、バックグラウンドでデータベース・サーバ自体 (5 ～ 9 行目) が起動されます。dbspawn.exe のサービスがないと、サーバ・プログラム dbsrv11.exe がフォアグラウンドで実行され、サーバがシャットダウンされるまで制御をコマンド・ファイルに戻さないため、これはコマンド・ファイルの作成時に役立ちます。言い換えれば、dbspawn がない状態では、サーバがシャットダウンされるまで、3 番目のコマンド (dbisql.com) は実行されません。

4 行目の -f パラメータは、他の SQL Anywhere サーバが実行されている可能性を無視し、この SQL Anywhere サーバを開始するよう dbspawn に指示します。

5 行目は、SQL Anywhere データベース・サーバ のネットワーク・バージョン dbsrv11.exe の完全なファイル仕様です。または、次に示す違いを除いてまったく同じ機能をサポートしている「パーソナル・サーバ」dbeng11.exe を使用することもできます。

- dbeng11.exe は最大 10 の同時接続をサポート
- 要求処理に最大 1 つの CPU を使用
- ネットワーク・クライアント／サーバ接続をサポートしていない

6 行目では、インメモリ書き込み禁止を示す "-im nw" オプションを指定しています。これは、最も効率的で、最も極端なインメモリ操作です。他のバージョン (インメモリ･チェックポイントのみを示す -im c) に伴うトランザクション・ログやテンポラリ・ファイルがないだけでなく、チェックポイント・ログも

なく、変更がデータベース・ファイルに書き込まれることはありません。

インメモリ操作は非常に大きな影響を及ぼします。すべてのデータがキャッシュに収まる必要があり、すべてのテンポラリ・データについても同様です。また、余剰分を収容するテンポラリ・ファイルもありません。さらに、後で必要になる場合に備えてデータを保存する責任はユーザに委ねられます。そのための SQL コードを作成したり、dbunload.exe ユーティリティを使用することも可能ですが、BACKUP 文も dbbackup.exe ユーティリティも役割を果たしません。機能はしても役に立たず、データをメモリにバックアップすることもなく、単に空のデータベース・ファイルになるだけです。

7 行目はオプションですが、ここで説明するテストでは、指定することが推奨されます。-c 1200M オプションは、初期データベース・キャッシュ・サイズを 1200 メガバイトに指定します。テストでは、このサイズがこの記事で使用したデータ・セットに適していることが示されました。ここで、「初期キャッシュ・サイズの選択」について詳しく説明しておきます。

> SQL Anywhere は、セルフ管理型データベースとして高く評価されています。多くのパフォーマンスおよび調整オプションのデフォルト値は厳選されており、多くの状況で、SQL Anywhere は初期設定のままで十分な力を発揮します。データベース・キャッシュ・サイズも例外ではありません。初期キャッシュ・サイズは、多くの環境において適切な結果を導き出す式によって自動的に計算され、キャッシュは、実行中も必要性の変化に応じて動的にサイズ変更されます。
>
> ただし、インメモリ・モードで空のデータベースを起動し、その後大量のデータをデータベース・キャッシュにロードする場合は、前述の「多くの状況」の例外となります。この場合、空のデータベース・ファイルは非常に小さいことから、初期キャッシュ・サイズを求める式によって 2 メガバイトが選択され、動的サイズ変更プロセスによってキャッシュ・サイズを 600 倍まで増加させる必要があります。これは、パフォーマンスに 2 つの悪影響を及ぼします。1 つは、最適でない内部キャッシュ構造がわずかな初期キャッシュ・サイズによって決定されること、もう 1 つは、キャッシュ・サイズの増加がデータのロードに伴って段階的に行われるため、ロード・プロセス自体が低速になる可能性があることです。
>
> このように空の状態から始まるインメモリ・データベースの場合、たとえば -c 500M、さらに -c 1G のように初期キャッシュ・サイズを推測するか、あるいはこのテストと同様に -c 1200M と指定することで、パフォーマンスを高めることができます。

図 4 の8 行目はオプションですが、すべての実働データベースでは指定することをおすすめします。-o パラメータは、データベース・サーバによって書き込まれるすべての診断メッセージのコピーを受け取るテキスト・ファイルのファイル仕様を指定します。この出力は一般に「コンソール・ログ」と呼ばれますが、トランザクション・ログと混同しないよう注意してください。SQL Server とは異なり、SQL Anywhere は、ユーザが -o ファイル仕様を指定しないかぎり、これらのメッセージをどこにも保存しません。

9 行目では、データベース・ファイルのファイル仕様(このテストでは mem.db) を指定しています。他のパラメータにより上書きされないかぎり、接続文字列 ENG=mem;DBN=mem; で使用されるランタイム「エンジン名」と「データベース名」がファイル名部分によって決まります。

11 行目および 12 行目では、この記事で扱う SQL 文の多くを実行するための Interactive SQL ユーティリティを起動しています。-c 接続文字列では、ランタイム・サーバまたはエンジン名 ENG=mem、ランタイム・データベース名 DBN=mem、SQL Anywhere ユーザ ID UID=dba、パスワード PWD=sql、およびオプションの接続名 CON=mem-1 を指定しています。

> ヒント：一般に SQL Anywhere データベースは分離され、ソフトウェアの他の層の背後に埋め込まれることから、同じユーザ ID がすべてのデータベース接続に使用され、問題をデバッグする際に接続を区別することが困難になる場合があります。この状況は、異なる CON= 接続名を使用することにより軽減できます。

図 5 は、図 3 で定義した SQL Server テーブルの SQL Anywhere バージョンです。SQL Anywhere は高度な Transact SQL 互換性を備えているため、構文の違いは、図 5 の 63 行目、67 行目、および 71 行目で定義している 3 つの計算済み カラムのみです。

図 5 : SQL Anywhere ターゲット・テーブルの作成 | コンテキスト : SQL Anywhere

```
1  CREATE TABLE sa_target (
2    sampling_id BIGINT NOT NULL,
3    ... lines 3 to 60 omitted, identical to Figure 3 ...
61   Userid VARCHAR ( 128 ) NULL,
62   previous_ApproximateCPUTime DECIMAL ( 30, 6 ) NOT NULL DEFAULT 0.0,
63   interval_ApproximateCPUTime DECIMAL ( 30, 6 ) NOT NULL COMPUTE (
64   COALESCE ( "ApproximateCPUTime", 0 )
65   - previous_ApproximateCPUTime ),
66   previous_Commit BIGINT NOT NULL DEFAULT 0,
67   interval_Commit BIGINT NOT NULL COMPUTE (
68   COALESCE ( "Commit", 0 )
69   - previous_Commit ),
70   previous_Rlbk BIGINT NOT NULL DEFAULT 0,
71   interval_Rlbk BIGINT NOT NULL COMPUTE (
72   COALESCE ( Rlbk, 0 )
73   - previous_Rlbk ),
74   PRIMARY KEY ( sample_set_number, connection_number ) );
```

手法 1 : BCP および LOAD TABLE によるダウンロード

この手法は 2 つの手順からなり、BCP (Bulk Copy Program : 一括コピー・プログラム) を使用してSQL Server ソース・テーブルをテキスト・ファイルにアンロードし、SQL Anywhere の LOAD TABLE 文によってそのファイルからインメモリ・データベース内のターゲット・テーブルにデータをロードします。

BCP ユーティリティは、SQL Server データベースとの間で大容量データをロードおよびアンロードするために広く利用されている高パフォーマンス・ツールです。

SQL Anywhere 側では、同様の機能が LOAD TABLE および UNLOAD SQL 文によって提供されます。

図 6 は、最初の手順である、BCP を実行する Windows コマンドラインです。

図 6 : BCP によるテキスト・ファイルへのアンロード SELECT | コンテキスト : SQL Server

```
1  "c:¥Program Files¥Microsoft SQL Server¥100¥Tools¥Binn¥bcp.exe"^
2  "SELECT * FROM main.dbo.mss_source"^
3  queryout "¥¥PAVILION2¥C¥data¥main¥mss_source.txt"^
4  -c^
5  -t $#$¥t^
6  -r $#$¥n^
7  -P j68Fje9#fyu489^
8  -S BRECK-PC¥TSUNAMI^
9  -U sa
```

図 6 の 1 行目は、Windows への標準 SQL Server 2008 セットアップによりインストールされる bcp.exe の完全なファイル仕様です。

2 行目では、“dbo” が所有する、データベース “main” 内のアンロード対象テーブル mss_source に含まれるすべてのローおよびカラムを選択します。SELECT のコーディングに関しては、BCP にはビット制限があり、「二重引用符」で囲み、すべてを 1 行に収める必要があります。継続文字を使用しても複数の行にまたがって記述することはできません。

3 行目では、(入力を表す “in” に対して) 出力を表す “queryout” を指定し、完全な出力ファイル仕様を指定しています。このファイル仕様は、BCP ユーティリティを実行しているコンピュータではなく、SQL Server 自体を実行しているコンピュータに対する相対パスです。このテストでは、BCP はサーバ・コンピュータではなくターゲット・ラップトップ・コンピュータで実行され、テキスト・ファイルはラップトップに書き込まれますが、重要なのは、BCP 自体がどこで実行されているかではなく、サーバに対するターゲット・ファイルの場所です。ファイル仕様が “¥¥server¥share¥...” 汎用命名規則形式に従って記述されているのはそのためで、これにより SQL Server はネットワークを介してデータを PAVILION2 コンピュータに書き込むことができます。

4 行目では、テキスト・ファイルに書き込まれるデータとして「文字」形式を使用することを表す -c を指定しています。「ネイティブ」形式を表す -n を指定することもできますが、これは他の SQL Server データベースに格納されたターゲット・テーブル用であり、SQL Anywhere で使用することはできません。

5 行目では、出力テキスト・ファイルで各カラム値の間に挿入する文字列を「-t 文字列」形式で指定しています。ドキュメントでは、これを「フィールド終端文字列」と呼んでいますが、各行で最終カラム値の後には挿入されないことから、実質的には区切り文字列です。デフォルトの区切り文字列は単一のタブ文字 (16 進数 09、または SQL Server 言語では ¥t) ですが、データにはタブ文字も含まれるため、終端文字列として別の文字列を選択する必要があります。-t 文字列 $#$¥t は、ドル記号、数値記号、ドル記号、およびタブを表し、後に示す LOAD TABLE 文内の SQL Anywhere の DELIMITED BY ‘$#$¥x09’ 句に相当します。

6 行目では、出力テキスト・ファイルでデータの各ローの末尾に挿入する「行終端文字列」を「–r 文字列」形式で指定しています。デフォルトは「改行」文字で、Windows のキャリッジ・リターンとライン・フィードのペア (CR-LF または 16 進数 0D0A) に相当します。この場合も、データには CR-LF などあらゆる種類の特殊文字も含まれるため、別の -r 文字列が指定されています。$#$¥n は、ドル記号、数値記号、ドル記号、および CR-LF を表します。これは、SQL Anywhere の ROW DELIMITED BY ‘$#$¥x0D¥x0A’ 句に相当します。

7、8、および 9 行目では、BCP から SQL Server への接続に使用するパスワード、サーバ、およびユーザ ID を指定しています。

図 7 に、すべてのローとカラムが必要な場合に、SELECT ではなくテーブル名を指定するよう図 6 の 2 行目および 3 行目を変更する方法を示します。このテーブル名構文はより単純ですが柔軟性に劣り、3 行目の

```
1 “c:¥Program Files¥Microsoft SQL Server¥100¥Tools¥Binn¥bcp.exe”^
2 main.dbo.mss_source^
3 out “¥¥PAVILION2¥C¥data¥main¥mss_source.txt”^
4 -c^
5 -t $#$¥t^
6 -r $#$¥n^
7 -P j68Fje9#fyu489^
8 -S BRECK-PC¥TSUNAMI^
9 -U sa
```

図 8 は、図 6 に示した BCP 手順により、190 万ローすべてが約 7.4 分でアンロードされたことを示しています。出力テキスト・ファイルのサイズは約 985M でした。

図 8 : BCP によるアンロードの画面出力 | コンテキスト : SQL Server

```
Starting copy...
1000 rows successfully bulk-copied to host-file. Total received: 1000
1000 rows successfully bulk-copied to host-file. Total received: 2000
1000 rows successfully bulk-copied to host-file. Total received: 3000
1000 rows successfully bulk-copied to host-file. Total received: 4000
---
1000 rows successfully bulk-copied to host-file. Total received: 1920000
1000 rows successfully bulk-copied to host-file. Total received: 1921000
1000 rows successfully bulk-copied to host-file. Total received: 1922000
1000 rows successfully bulk-copied to host-file. Total received: 1923000
1000 rows successfully bulk-copied to host-file. Total received: 1924000
1000 rows successfully bulk-copied to host-file. Total received: 1925000
1925469 rows copied.
Network packet size (bytes): 4096
Clock Time (ms.) Total : 446469 Average : (4312.66 rows per sec.)
```

図 9 は、図 6 で示した BCP により作成されたファイルをロードする SQL Anywhere コマンドです。この手順では、約 4.6 分で 190 万ローすべてがメモリにロードされました。前の項で説明したとおり、図 4 の SQL Anywhere サーバ・コマンドラインで初期キャッシュ・サイズ・パラメータ -c 1200M を省略した場合、図 9 の LOAD TABLE は、5 分未満から 11 分超へと大幅に低速になります。

図 9 : SQL Anywhere の LOAD TABLE | コンテキスト : SQL Anywhere

```
1 LOAD TABLE sa_target
2   FROM 'c:/data/main/mss_source.txt'
3   DELIMITED BY '$#$¥x09'
4   ESCAPES OFF
5   QUOTES OFF
6   ROW DELIMITED BY '$#$¥x0D¥x0A';
```

図 9 の 1 行目では、ロードするターゲット・テーブル名を指定し、2 行目では、入力ファイルのファイル仕様を指定しています。

ヒント : 文字列リテラル中のフォワード・スラッシュ "/" は、SQL Anywhere が Windows 上でファイル仕様を処理する際にバックスラッシュ "¥" として解釈されます。フォワード・スラッシュを使用することにより、SQL Anywhere において文字列リテラル内でバックスラッシュがエスケープ文字として解釈される状況との混同を避けることができます。すなわち、通常、¥n は改行文字として解釈されますが、/n は改行文字として解釈されません。"/" が "¥" として解釈されるのはファイル仕様にのみ適用される特殊なトリックで、その他のコンテキストでは、フォワード・スラッシュはフォワード・スラッシュとして扱われます。

図 9 の3 行目は、カラム区切り文字列の SQL Anywhere の構文で、これは図 6 の 5 行目で示した SQL Server 構文と一致しており、ドル記号、数値記号、およびドル記号の後に、16 進数エスケープ文字列 ¥x09 で表されたタブ文字が続いています。

4 行目では、SQL Server がバックスラッシュ文字を使用して特殊文字をエスケープ文字列に変換しないことを SQL Anywhere に通知しています。たとえば、入力文字列にタブ文字が含まれている場合、16 進数エスケープ文字列 ¥x09 ではなく実際の単一のタブ文字が含まれます。

5 行目では、SQL Server が文字列値を「一重引用符」または「二重引用符」で囲まないことを SQL Anywhere に通知しています。

6 行目は、ロー終端文字列の SQL Anywhere の構文で、図 6 の 6 行目に示した SQL Server 構文に一致しており、ドル記号、数値記号、ドル記号の後に、16 進数エスケープ文字列 ¥x0D および ¥x0A で表したキャリッジ・リターンが続いています。

手法 2： Mobile Link によるダウンロード

Mobile Link は、SQL Anywhere に同梱されている双方向同期ソフトウェアです。1 つ以上の SQL Anywhere「リモート」データベースと、SQL Server、Oracle、IBM DB2、Sybase ASE、SQL Anywhere、または MySQL で実行されている 1 つの中央「統合」データベースと連携します。

Mobile Link は、クライアント・コンポーネントとサーバ・コンポーネントで構成され、各コンポーネントはそれぞれリモート・データベースと統合データベースに接続します。この項で説明する手法では、SQL Server 統合データベースから SQL Anywhere リモート・データベースへの一方向 (ダウンロード) にのみ Mobile Link を使用します。このプロセスは次のように動作します。

a) Mobile Link サーバは、一般には SQL Server を実行中のコンピュータ上で実行されているサービス、またはそのコンピュータに近いサービスとして開始されると、SQL Server 統合データベースへの ODBC 接続を確立し、Mobile Link クライアントからのアクセスを待機します。

b) Mobile Link クライアントは、同期セッションを開始するために起動すると、SQL Anywhere リモート・データベースへのデータベース接続を確立し、さらに専用の高レベル・プロトコルを使用して Mobile Link サーバへのネットワーク接続を確立します。

c) Mobile Link サーバは、ユーザ記述の SQL SELECT 文を SQL Server データベースに対して実行し、結果セットを専用のネットワーク・プロトコル経由で Mobile Link クライアントに送信します。

d) Mobile Link クライアントは、SQL INSERT 文を使用して、ダウンロードした結果セットを 1 ローずつ SQL Anywhere データベースに適用します。

Mobile Link についてご存知の方は、上記の手順が Mobile Link に組み込まれている機能と柔軟性を活用することから始まっていないことにお気付きでしょう。インメモリ・データベースへのデータのダウンロードに関しては、次の機能が役立つことがあります (ただし、この記事では扱わない)。

a) ローの挿入または最終更新の日時を記録する日時カラムがローに含まれている場合、ダウンロード SELECT の WHERE 句に、前回の同期以降に挿入または更新されたローを選択する述部を含めることができます。Mobile Link は、この目的のために、正常に実行された最後の同期の日時を自動的に維持します。インメモリ・データベース内のデータが継続的に保存されている場合、すなわち -im nw の代わりに -im c オプションが使用されている場合 (書き込み禁止の代わりにチェックポイントが指定されたインメモリ)、この手法がさらに役立つことがあります。

b) 今年のデータのみをダウンロードするなど、ローのサブセットを選択するために、その他の述部を使用することもできます。

c) 結果セットが、リモート・データベースで定義されているロー・レイアウトと一致していれば、さらにストアド・プロシージャ・コールを使用して、必要に応じてダウンロード SELECT 文を複合文にすることができます。そのため、統合データベースとリモート・データベース、正規化デザインと非正規化デザインなどスキーマの違いに容易に対応することができます。

図 10 に、同期の実行前に、Mobile Link システム・テーブルとその他の Mobile Link スキーマ・オブジェクトを SQL Server データベースに追加する方法を示します。Mobile Link には、ユーザ記述の構成ファイルは必要ありません。サーバ側の同期を実行するために必要なものは、すべて SQL Server データベース自体に格納されています。

図 10： SQL Server での Mobile Link システムのセットアップ | コンテキスト： SQL Server

```
1 "C:¥Program Files¥Microsoft SQL Server¥100¥Tools¥Binn¥sqlcmd.exe"^
2  -i "%SQLANY11%¥MobiLink¥setup¥syncmss.sql"^
3  -d main^
4  -I^
5  -P j68Fje9#fyu489^
6  -S BRECK-PC¥TSUNAMI^
7  -U sa
```

図 10 の 2 行目では、標準 SQL Anywhere セットアップ中にインストールされる入力 SQL コマンド・ファイルを指定しています。このファイルには、前述のダウンロード SELECT 文を格納するために Mobile Link が使用する ml_script テーブルの CREATE 文が含まれています。Mobile Link サーバは、同期セッションを開始すると、ml_script から SELECT 文を読み取った後に方向転換し、その文を SQL Server に対して実行します。syncmss.sql では、これ以外にも多数のオブジェクトが定義されていますが、ml_script はここで説明する手法の心臓部となるものです。

図 11 に、ダウンロード SELECT を SQL Server 上の Mobile Link システム・テーブルに追加する方法を示します。

図 11： SQL Server 上の Mobile Link ダウンロード・スクリプト | コンテキスト： SQL Server

```
1 USE main
2 GO
3
4 EXECUTE ml_add_table_script
5  'v1',
6  'sa_target',
7  'download_cursor',
8  'SELECT * FROM mss_source'
9 GO
```

図 11 の 4 行目では、図 10 に示したコマンドによって作成された Mobile Link システム・プロシージャの 1 つを起動しています。このテストでは、ml_add_table_script プロシージャにより、新しいテーブルに固有の Mobile Link スクリプトを ml_script テーブルに追加しています。

5 ～ 7 行目では、8 行目の SELECT 文を一意に識別する、3 つの要素からなるプライマリ・キーを指定しています。

5 行目のスクリプト・バージョン 'v1' を使用することで、さまざまなバージョンのアプリケーションで使用されるあらゆる Mobile Link スクリプトの定義が容易になります。この記事では、1 つのバージョンの単一スクリプトのみを使用するため、バージョンは重要ではありません。

6 行目では、このスクリプトが適用されるリモート・データベース・テーブル名 'sa_target' を指定しています。8 行目の SELECT * FROM mss_source により生成される SQL Server 結果セットは、SQL Anywhere データベース上の sa_target に送信されます。

7 行目では、どの「Mobile Link イベント」によって 8 行目のスクリプトを起動するかを指定しています。Mobile Link サーバはイベント駆動型で、数十種類の接続、同期、およびテーブルレベルのイベントがあります。複合同期アプリケーションでは、認証、競合解決、およびアップロード処理や、ダウンロード SELECT 文などさまざまな目的のために数百もの同期スクリプトが必要になります。

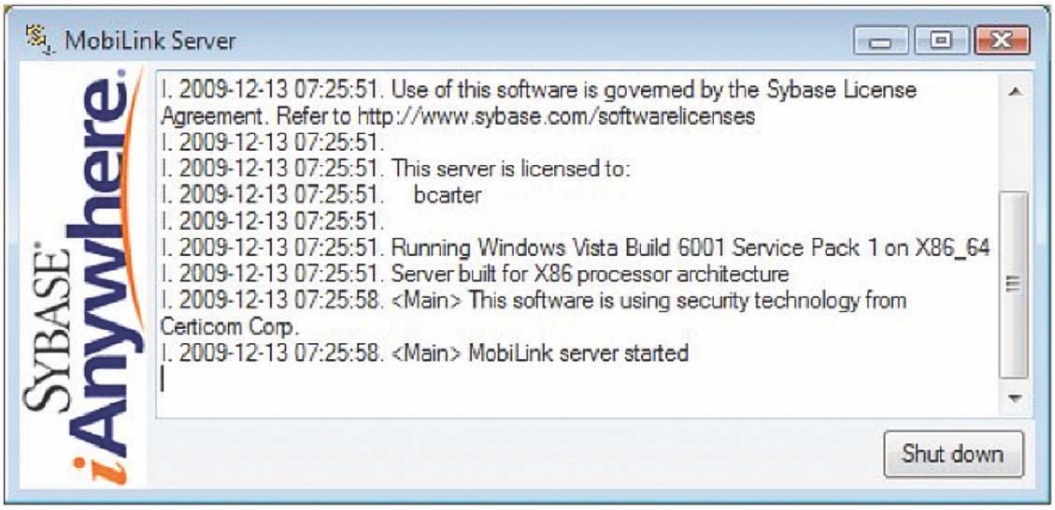
注： Mobile Link システム・テーブルの実際のスキーマは、ここで暗示しているものより多少複雑です。
そのため、テーブル・レイアウトを調べてみると、ml_script に単一カラムの代理プライマリ・キーがあり、リレーションシップ・テーブル ml_table_script に前述の 3 つの要素からなるプライマリ・キーが含まれており、さらに多くのリレーションシップ・テーブルにより、バージョンおよびテーブルを識別するために整数が使用されていることがわかります。ただし、ここでの説明ではどれもそれほど重要ではありません。

この記事では、Mobile Link イベントとして、テーブル・レベルの 'download_cursor' スクリプトのみを使用します。これは、実際にコーディングする SELECT、または結果セットを返すプロシージャ・コールであり、決してカーソル定義ではないため、理想としてはこのイベントを 'download_select' とするべきでしょう。

Mobile Link サーバは、同期プロセスにおいて、データを sa_target テーブルにダウンロードする時点に達すると、sa_target の download_cursor イベントをトリガーし、図 11 の 8 行目のスクリプトが実行されます。

図 12 は、サーバ・コンピュータ BRECK-PC 上で実行されている Mobile Link サーバ、図 13 は、 Mobile Link サーバを開始するために使用したコマンドです。この時点では、Mobile Link サーバはクライアント側からの同期要求を待機しています。

図 12： Mobile Link サーバのウィンドウ | コンテキスト： SQL Server




```
1  "%SQLANY11%¥bin32¥mlsrv11.exe"^
2    -c  "DSN=main_BRECK-PC;UID=sa;PWD=j68Fje9#fyu489"^
3    -o  mlsrv11_log.txt^
4    -vscn^
5    -zu+
```

図 13 の 1 行目は、Mobile Link サーバ mlsrv11 のファイル仕様です。2 行目では、Mobile Link サーバが SQL Server データベースに接続するために使用する ODBC データベース接続文字列、すなわち、ODBC DSN と、SQL Server のユーザ ID およびパスワードを指定しています。

3 行目では、Mobile Link サーバが診断および進捗メッセージを書き込むテキスト・ファイルを指定しています。4 行目の -vscn オプションは、冗長レベル、すなわちスクリプト名、スクリプトの内容、およびロー数を設定します。実働環境でも、ネットワークが関与するたびに何らかの事柄が発生するため、経験上、この診断ファイルは、開発および実働の両方におけるデバッグ・プロセスに不可欠です。

5 行目の -zu+ オプションでは、Mobile Link クライアントから送信された Mobile Link ユーザ名の認証に介入しないよう Mobile Link サーバに指示します。Mobile Link クライアントとサーバ間のセキュリティを強化する必要がある場合は、Mobile Link ユーザ名とパスワードを事前に定義し、このオプションを省略して強制的に認証を行うようにすることができます。また、Mobile Link クライアントとサーバ間のパスに対してトランスポート層およびエンドツーエンドの暗号化を指定することも可能ですが、暗号化の詳細についてはこの記事では扱いません。

図 14 は、サーバ・コンピュータのレジストリに格納されているユーザ DSN です。

図 14 : SQL Server データベースの ODBC ユーザ DSN | コンテキスト : SQL Server

```
1  Windows Registry Editor Version 5.00
2
3  [HKEY_CURRENT_USER¥Software¥ODBC¥ODBC.INI¥main_BRECK-PC]
4  “Driver”=”C:¥¥Windows¥¥system32¥¥sqlncli10.dll”
5  “Server”=”BRECK-PC¥¥TSUNAMI”
6  “Database”=”main”
7  “LastUser”=”sa”
8
9  [HKEY_CURRENT_USER¥Software¥ODBC¥ODBC.INI¥ODBC Data Sources]
10 “main_BRECK-PC”=”SQL Server Native Client 10.0
```

図 15 は、クライアント側の SQL Anywhere リモート・データベースに必要な Mobile Link セットアップです。Mobile Link では、「パブリケーション」と同様に同期対象のテーブルが指定され、Mobile Link「ユーザ」が「サブスクリプション」によってそのパブリケーションにリンクされる、簡略化された「パブリッシュ・サブスクライブ」モデルが使用されます。

図 15 : SQL Anywhere 上の Mobile Link セットアップ | コンテキスト : SQL Anywhere

```
1  CREATE PUBLICATION p1 FOR DOWNLOAD ONLY
2   ( TABLE sa_target );
3
4  CREATE SYNCHRONIZATION USER “1” TYPE tcpip;
5
6  CREATE SYNCHRONIZATION SUBSCRIPTION TO p1 FOR “1”;
```

図 15 の 1 行目および 2 行目では、単一のテーブル sa_target で構成される p1 というパブリケーションを定義しています。さらに複雑なセットアップでは、カラムのサブセットを指定するカラム名リストとアップロード対象のローを制御する WHERE 句が含まれる複数のテーブル名を使用できます。

通常、テーブルに対してアップロードおよびダウンロードが行われますが、FOR DOWNLOAD ONLY 句によりこの動作が変更されています。このようにした理由は 2 つあります。1 つは、ここで説明する手法がダウンロードのみであることです。もう 1 つは、インメモリ・データベースにはトランザクション・ログが存在しないため、通常の Mobile Link のトランザクション・ログ駆動のアップロード・プロセスが不可能なことです。Mobile Link では、非ログベースのスクリプト駆動型アップロードも可能ですが、ここではアップロードは必要ありません。

4 行目では、Mobile Link ユーザ ID として “1” を指定し、Mobile Link クライアントとサーバ間のネットワーク・トランスポート層として TCP/IP を指定しています。このような単純なセットアップでは、Mobile Link ユーザ ID によって、リモート SQL Anywhere データベースが一意に識別されます。その他のクライアントでは、2、3 など、またはその他一意の VARCHAR (128) 文字列を使用します。必要なのはこのユーザ ID のみで、これによって Mobile Link サーバで複数の同期セッションが区別されます。

注 : Mobile Link ユーザ ID は、ユーザが管理する、グローバルに一意の効果的なリモート・データベース識別子です。データベース・ユーザ ID ではありません。

6 行目では、「サブスクリプション」を作成し、ユーザ ID 1 をパブリケーション p1 にサブスクライブしています。

図 16 は、ラップトップ・コンピュータ PAVILION2 上で実行されている Mobile Link クライアントです。図 17 は、Mobile Link クライアントを開始するために使用したコマンドです。この時点では、Mobile Link サーバが応答し、同期が進行中です。実際、すべてのデータが Mobile Link クライアントに送信され、130 万ローが SQL Anywhere データベースに挿入されています。

図 16： Mobile Link クライアントのウィンドウ | コンテキスト： SQL Anywhere

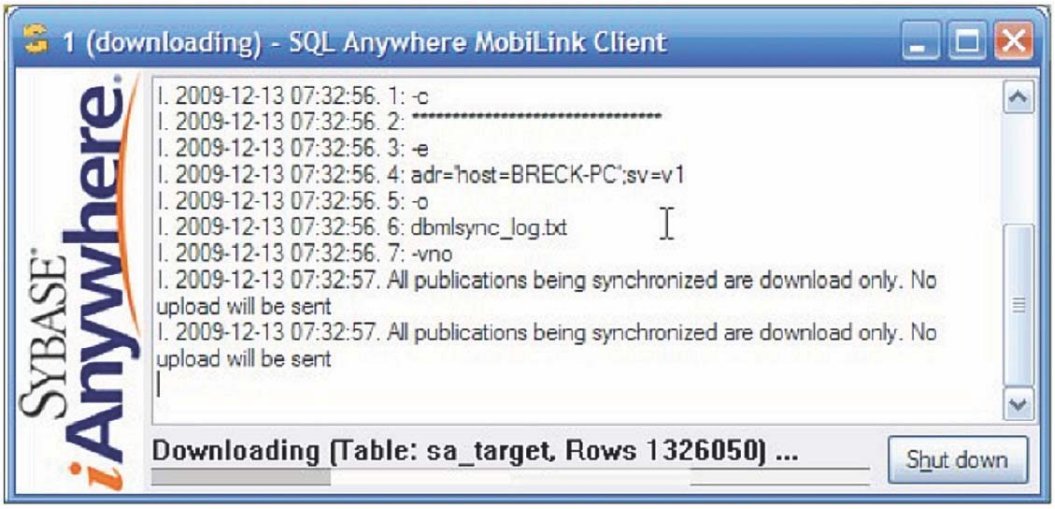


図 17： Mobile Link クライアントによる同期セッションの実行 | コンテキスト： SQL Anywhere

```
1  "%Sqlany11%¥Bin32¥Dbmlsync.Exe"^
2    -C   "Eng=Mem;Dbn=Mem;Uid=Dba;Pwd=Sql"^
3    -E   "Adr='Host=Breck-Pc';Sv=V1"^
4    -O   Dbmlsync_log.Txt^
5    -Vno
```

図 17 の 1 行目は、dbmlsync プログラムの完全なファイル仕様、2 行目は、サーバまたはエンジン名、データベース名、および SQL Anywhere ユーザ ID とパスワードで構成される標準 SQL Anywhere データベース接続文字列です。

3 行目では、追加の同期パラメータを指定しています。このテストでは、SQL Server とともに BRECK-PC コンピュータ上で実行されている Mobile Link サーバのホストアドレスと使用する Mobile Link スクリプト・セットを Mobile Link サーバに通知するスクリプト・バージョン sv=v1 を指定しています。

4 行目では、Mobile Link クライアントが診断および進捗メッセージを書き込むテキスト・ファイルを指定し、5 行目では、-vno によって、ロー数および使用オプションを含めるようこれらのメッセージの冗長性を設定しています。クライアント側の診断ファイルは、図 13 の 3 行目で定義したサーバ側のファイルほど便利ではありませんが、役立つこともあります。

手法 3：プロキシ・テーブルによるダウンロード

リモート・データ・アクセスは、10 年以上前に SQL Anywhere バージョン 6 に導入されました。この機能では、実際に SQL Anywhere データベース内にデータを格納しなくても、そのデータベース内のテーブルのスキーマを作成できます。これは、Excel スプレッドシート、Oracle または DB2 データベース、テキスト・ファイル、あるいはこの記事で使用する SQL Server データベースなど、他の場所に実際に格納されるデータを表すことから、「プロキシ・テーブル」と呼ばれます。

図 18 : プロキシ・テーブルによるダウンロード | コンテキスト： SQL Anywhere

```
1  CREATE SERVER mss
2    CLASS 'MSSODBC'
3    USING 'DSN=main_BRECK-PC';
4
5  CREATE EXTERNLOGIN DBA
6    TO mss
7    REMOTE LOGIN "sa"
8    IDENTIFIED BY 'j68Fje9#fyu489';
9
10 CREATE EXISTING TABLE proxy_mss_source
11   AT 'mss.main.dbo.mss_source';
12
13 INSERT sa_target
14 SELECT *
15   FROM proxy_mss_source;
```

図 18 の 1 行目では、SQL Anywhere データベース内にリモート・サーバ・スキーマ・オブジェクトを作成し、それにローカル名 mss を付けます。

2 行目では、使用するアクセス・パスまたはサーバ「クラス」を指定しています。現時点で SQL Anywhereで使用できるクラスは、Excel やテキスト・ファイルなどのための vanilla ODBC クラスから、ODBC による Oracle および JDBC による Sybase ASE まで 12 種類あります。このテストには、ODBC による MSS クラスである MSSODBC が最善の選択肢です。

3 行目では、使用する ODBC DSN を指定しています。USING 句で ODBC ドライバ情報を記述することにより、DSN レス接続を指定することも可能ですが、このテストでは図 19 に示した DSN を使用します。

5 〜 8 行目では、SQL Anywhere がバックグラウンドで SQL Server に接続するために使用する「外部ログイン」パスを定義しています。前に mss という名前を付けたリモート・サーバへの接続には、ローカル・ユーザ ID DBA と、REMOTE LOGIN 句および IDENTIFIED BY 句で与えられる SQL Server ユーザ ID およびパスワードが使用されます。

10 行目で、proxy_mss_source というプロキシ・テーブルの定義を開始しています。EXISTING 句は、実際のテーブルが SQL Server 上にすでに存在するため作成する必要はなく、SQL Anywhere 上にプロキシ・テーブルを作成しさえすればよいことを指定します。

11 行目の AT 句では、SQL Server 上の実際のテーブルを指定しています。最初のパラメータであるリモート・サーバ名 mss により、CREATE EXTERNLOGIN ログイン・パスを前提に CREATE SERVER アクセス・パスを指定します。その他の 3 つのパラメータでは、SQL Server データベース main、テーブル所有者 dbo、および実際のテーブル mss_source を指定しています。

13 〜 15 行目は、proxy_mss_source テーブルを介して mss_source のすべてのローを取得し、sa_target という実際の SQL Anywhere テーブルに挿入する方法を示しています。前述の 2 つの手法 (LOAD TABLE および Mobile Link) と同様に、図 18 の INSERT は単一のトランザクションとして実行され、SQL Anywhere 内で 190 万のロー・ロックを収集し、その後解放します。

図 19は、ラップトップ・コンピュータのレジストリに格納されているユーザ DSN です。別のコンピュータのレジストリに格納されていることを除けば、この DSN は前の項の Mobile Link で使用したものとまったく同じです。図 14 を参照してください。

図 19 : プロキシ・テーブルの DBC ユーザ DSN | コンテキスト： SQL Server

```
1  Windows Registry Editor Version 5.00
2
3  [HKEY_CURRENT_USER¥Software¥ODBC¥ODBC.INI¥main_BRECK-PC]
4  “Driver”=”C:¥¥Windows¥¥system32¥¥sqlncli10.dll”
5  “Server”=”BRECK-PC¥¥TSUNAMI”
6  “Database”=”main”
7  “LastUser”=”sa”
8
9  [HKEY_CURRENT_USER¥Software¥ODBC¥ODBC.INI¥ODBC Data Sources]
10 “main_BRECK-PC”=”SQL Server Native Client 10.0”
```

手法 4 : リンク・サーバによるダウンロード

リンク・サーバ機能は、SQL Server でのアドホック・クエリによる、OLEDB および ODBC 経由でのさまざまな外部データ・ソース操作を可能にするために SQL Server 2000 に導入されました。リンク・サーバで提供される機能は SQL Anywhere のプロキシ・テーブルと同様ですが、構文が異なります。すなわち、プロキシ・テーブルを参照する代わりに、SQL Server で実行されている SQL 文から他の場所にある実際のテーブルを直接参照します。

プロキシ・テーブルと同様に、リンク・サーバを使用するために必要なのは、実際のデータへの ODBC アクセスと数行の SQL コードのみです。図 20 に、SQL Server が SQL Anywhere に接続するためのリンク・サーバを定義し、INSERT SELECT を実行してローカル・テーブル mss_source からターゲット・テーブル sa_target にすべてのデータをコピーするために必要なコードを示します。

```
1  EXEC sp_addlinkedserver
2    @server = ‘mem’,
3    @srvproduct = ‘xxx’,
4    @provider = ‘MSDASQL’,
5    @datasrc = ‘sa_system_dsn’
6  GO
7
8  EXEC sp_addlinkedsrvlogin
9    @rmtsrvname = ‘mem’,
10   @useself = ‘false’,
11   @rmtuser = ‘dba’,
12   @rmtpassword = ‘sql’
13 GO
14
15 INSERT INTO mem..dba.sa_target
16 SELECT *
17   FROM mss_source
18 GO
```

図 20 の 1 ～ 6 行目では、SQL Server にリンク・サーバを定義するシステム・ストアド・プロシージャを呼び出しています。

2 行目では、後に SQL 文で使用する論理ローカル・サーバ名 “mem” を指定しています。

3 行目では、リンク・サーバの「製品名」を指定しています。一見、少なくともこのテストにおいては、デフォルトの NULL 値以外を指定すれば、このパラメータは重要ではないように思われます。

4 行目では、SQL Anywhere との通信に使用する OLE DB プロバイダを指定しています。このテストでは、ODBC の標準Microsoft OLE DB プロバイダである MSDASQL を指定しています。SQL Anywhere 11 に同梱されている SAOLEDB.11 プロバイダを使用することもできますが、MSDASQL プロバイダは、この記事で使用する単純な SQL を処理することができます。

5 行目では、OLE DB プロバイダが SQL Anywhere へのアクセスに使用する ODBC システム DSN を指定しています。この DSN のレジストリ・エントリを次の図 21 に示します。

図 20 の 8 ～ 13 行目では、SQL Server がバックグラウンドで SQL Anywhere への接続に使用するログイン・パスを定義するシステム・ストアド・プロシージャを呼び出しています。

9 行目では、前述の 2 行目で定義したリンク・サーバを指定しています。

10 行目は、SQL Anywhere への接続にローカル SQL Server ログインID を使用せず、SQL Anywhere に固有のユーザ ID とパスワードを 11 行目と 12 行目で指定することを示します。

15 ～ 18 行目は、すべてのローをローカル SQL Server テーブル mss_source から取得し、SQL Anywhere テーブル sa_target に挿入する方法を示しています。

15 行目では、4 つの要素からなる標準の SQL Server テーブル命名規則 [サーバ].[データベース].[所有者].[テーブル] を使用しています。このテストの場合、サーバは前に定義したリンク・サーバ mem です。SQL Anywhere では接続ごとにデータベースは 1 つのみであるため、データベース名は省略されています。最後の 2 つの要素は所有者とテーブル名 (dba.sa_target) です。

図 21 は、サーバ・コンピュータのレジストリに格納されているシステム DSN です。これは単純なSQL Anywhere DSN で、UID、PWD、DatabaseName、ServerName、および CommLinks 以外はすべてデフォルト値です。

図 21 : リンク・サーバの ODBC システム DSN | コンテキスト： SQL Anywhere

```
1  Windows Registry Editor Version 5.00
2
3  [HKEY_LOCAL_MACHINE¥SOFTWARE¥ODBC¥ODBC.INI¥sa_system_dsn]
4  “Driver”=”C:¥¥Program Files¥¥SQL Anywhere 11¥¥bin64¥¥dbodbc11.dll”
5  “UID”=”dba”
6  “PWD”=”sql”
7  “DatabaseName”=”mem”
8  “ServerName”=”mem”
9  “AutoStop”=”YES”
10 “Integrated”=”NO”
11 “CommLinks”=”TCPIP{host=PAVILION2}”
12 “Compress”=”NO”
```

手法 5： OPENROWSET によるダウンロード

OPENROWSET は、実際にリンク・サーバを定義しなくても、リンク・サーバの多くの機能を活用することのできる特殊な SQL Server 機能です。OPENROWSET の構文は多少粗削りですが、図 22 に示すようにコードが簡潔になります。

図 22： OPENROWSET によるダウンロード | コンテキスト： SQL Server

```
1  INSERT  INTO  OPENROWSET  (  'MSDASQL',
2   'sa_system_dsn'; 'dba';  'sql',
3   dba.sa_target  )
4  SELECT  *
5   FROM mss_source
6  GO
```

図 22 の 1 行目は、SQL Server の INSERT 文のターゲットとして OPENROWSET 関数呼び出しを使用できることを示しています。OPENROWSET 呼び出しは、UPDATE および DELETE 文のターゲットにすることもでき、これらの文を SELECT 文の FROM 句に記述できます。

また、1 行目では、SQL Anywhere との通信に使用する OLE DB プロバイダも指定しています。このテストの場合は、前述のリンク・サーバによるダウンロード手法で使用したものと同じ標準 SQL Server ODBC プロバイダ MSDASQL です。実際、基礎となる技術に関するかぎり、OPENROWSET 手法はリンク・サーバ手法の変形です。

2 行目では、使用する ODBC DSN と、SQL Anywhere に渡すユーザ ID およびパスワードを指定しています。これについても、必要な情報はリンク・サーバの場合と同じです。粗削りの構文が力を発揮するのは、この 2 行目です。この行では、3 つのパラメータを、カンマではなくセミコロンで区切ります。

3 行目では、SQL Anywhere データベース上のターゲット・テーブルを指定しています。リンク・サーバは関与しないため、完全な [サーバ].[データベース].[所有者].[テーブル] 構文ではなく、より簡潔な [所有者].[テーブル] 命名規則が使用されています。

4 ～ 6 行目では、INSERT による処理のために、SQL Server ソース・テーブルのすべてのローを選択しています。

```
1  USE  master
2  GO
3
4  sp_configure  'show advanced options',  1
5  GO
6
7  RECONFIGURE
8  GO
9
10 sp_configure  'Ad  Hoc  Distributed  Queries',  1
11 GO
12
13 RECONFIGURE
14 GO
15
16 SELECT  *
17  FROM OPENROWSET  (  'MSDASQL',
```

```
18  'sa_system_dsn'; 'dba'; 'sql',
19  sys.dummy )
20  GO
21
22  dummy_col
23  ------------
24  0
```

図 23 の 10 行目は、セットアップの中心となる手順で、アドホック分散クエリ機能を有効にすることにより、OPENROWSET を可能にします。

16 ～ 20 行目は、セットアップが機能するかどうかを確認するための単純なテストで、SELECT により、SQL Anywhere の単一ロー、単一カラムの「ダミー」テーブルを表示します。OPENROWSET が正常に機能していれば、22 ～ 24 行目に示す結果が表示されます。表示されない場合は、"Msg 15281 SQL Server blocked access to STATEMENT 'OpenRowset/OpenDatasource'" (メッセージ 15281 SQL Server は 'OpenRowset/OpenDatasource' 文へのアクセスをブロックしました) といったメッセージが表示されます。

手法 4 および 5 の再検討： SAOLEDB11 プロバイダ

SQL Anywhere 11 には、SQL Server 用の、SAOLEDB.11 という代替 OLE DB プロバイダが同梱されています。

ヒント：理論上、「ドキュメント」とも呼ばれ、より単純な名前 "SAOLEDB" で SAOLEDB.11 を参照することもできますが、この名前を使用するとうまく機能せず、完全な名前 "SAOLEDB.11" でのみ成功することが判明しました。

注意：この項で説明する手法は、SQL Server により blob と見なされるカラムがテーブルに含まれていると、「SQL Server 致命的メモリ・リーク」とも呼ばれる状況に陥ります。それでは、「なぜ SAOLEDB.11 を使用するのか」というと、「MSDASQL より高速だから」です。これについては、次の「パフォーマンス」の項で詳しく説明します。

図 24 に、SQL Server を実行しているコンピュータに SAOLEDB.11 プロバイダを登録するために必要な Windows コマンドラインの 2 つのバージョンを示します。これらのバージョンの違いは、図 25 に示すようなダイアログ・ボックスがあるかないかです。

図 24 : SAOLEDB.11 プロバイダの登録 | コンテキスト： SQL Server

```
1  REM With "OK" dialog boxes...
2  regsvr32 dboledb11.dll
3  regsvr32 dboledba11.dll
4  PAUSE
5
6  REM Without "OK" dialog boxes...
7  regsvr32 /s dboledb11.dll
8  regsvr32 /s dboledba11.dll
9  PAUSE
```

図 25： RegSvr32 dboledb1 | コンテキスト： SQL Server

図 26 は、SQL Server が SAOLEDB.11 プロバイダを有効にするために必要な「魔法の設定」です。

注：図 26 の 7 ～ 10 行目の EXEC により、メッセージ “RegDeleteValue() returned error 2, ‘The system cannot find the file specified.’” (RegDeleteValue() がエラー 2「システムは、指定されたファイルを見つけることができません。」 を返しました) を生成します。多くの場合、これは SAOLEDB.11 について DisallowAdHocAccess がすでにゼロに設定されていることを意味するため、続行してもかまいません。

```
1  EXEC  master.dbo.sp_MSset_oledb_prop
2    N’SAOLEDB.11’,
3    N’AllowInProcess’,
4    1
5  GO
6
7  EXEC  master.dbo.sp_MSset_oledb_prop
8    N’SAOLEDB.11’,
9    N’DisallowAdHocAccess’,
10   0
11 GO
```

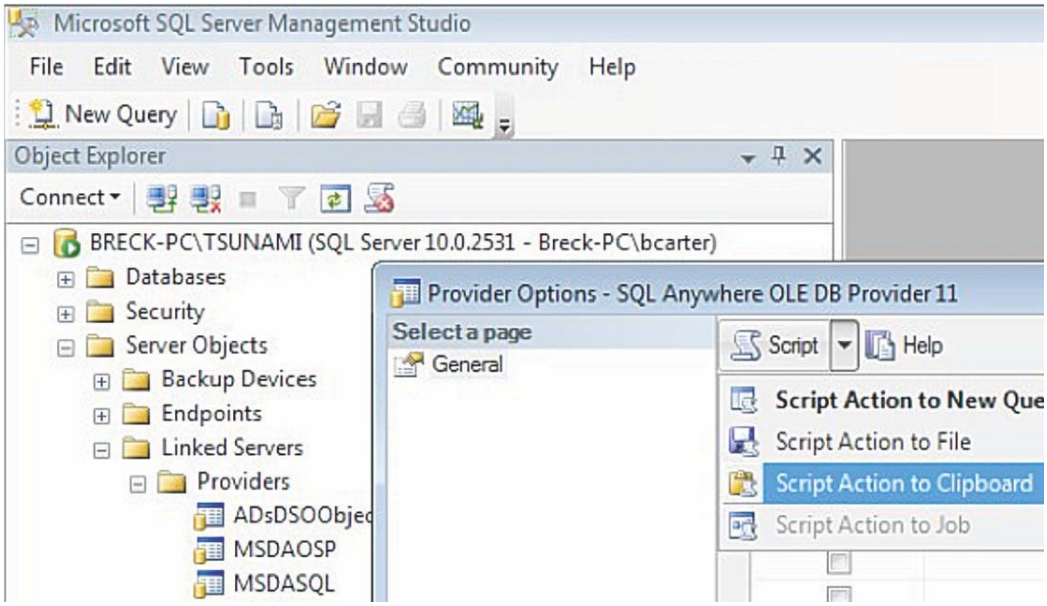
ここでヒントを示します。スクリプトを作成するために SQL Server Management Studio を入手し、多くのプロパティ・ダイアログ・ボックスに表示される [Script] メニュー項目を使用することができます。

たとえば、[Allow inprocess] を有効にするタスクについて考えます。これには、次の手順をたどる必要があります。

- Microsoft SQL Server Management Studio
- [<サーバ名>]
- [Server Objects]
- [Linked Servers]
- [Providers]
- SAOLEDB.11
- [Provider Options] — SQL Anywhere OLE DB Provider 11
- [General] ページ
- [Allow inprocess] チェックボックスをオン

[Allow inprocess] オプションをオンにし、[Script] - [Script Action to Clipboard] を選択すると、図 27 のような SAOLEDB.11 の [Provider Options] ダイアログが表示されます。

図 27 クリップボードへのスクリプトのコピー | コンテキスト： SQL Server



クリップボードにコピーされるコードは次のとおりです。このコードは、スクリプト・ファイル (多少編集されたバージョンは図 26 を参照) に貼り付けることができます。

```
1  USE [master]
2  GO
3  EXEC master.dbo.sp_MSset_oledb_prop N'SAOLEDB.11', N'AllowInProcess', 1
4  GO
```

これはまさに魔法のコードです。SQL Server 2008 のヘルプには、ストアド・プロシージャ sp_MSset_oledb_prop についての記述がどこにもありません。また、単独でのドキュメントもなく、私自身が探したかぎりでは Web 上にもドキュメントはありません。上記のようなコードの例がいくつかあるのみです。

もう 1 つのヒントとして、[Script Action to Clipboard] メニュー項目では、多少の変更を行わないかぎり、実際にはコードが生成されませんが、オプションを変更して後で元に戻すことは簡単なため、コードを取得することは可能です。

図 28 に、手法 4：リンク・サーバによるダウンロードを使用する場合に、MSDASQL の代わりに SAOLEDB.11 プロバイダを使用してリンク・サーバを作成する方法を示します。図 28 のコードは前述の図 20 によく似ていますが、3 行目の @srvproduct に実際の値を入力し、4 行目で @provider の新しい値を指定して、11 行目で @locallogin パラメータに NULL 値を割り当てる必要があります。

図 28： SAOLEDB.11 を使用したリンク・サーバの作成 | コンテキスト： SQL Server

```
1  EXEC sp_addlinkedserver
2    @server = 'mem',
3    @srvproduct = 'SQL Anywhere OLE DB Provider',
4    @provider = 'SAOLEDB.11',
5    @datasrc = 'sa_system_dsn'
6  GO
7
8  EXEC sp_addlinkedsrvlogin
9    @rmtsrvname = 'mem',
10   @useself = 'false',
11   @locallogin = NULL,
12   @rmtuser = 'dba',
13   @rmtpassword = 'sql'
14 GO
```

図 29 に、リンク・サーバが正しく設定されているかどうかを確認するための、簡易テストの実行方法を示します。SQL Anywhere 上の単一カラム、単一ローの「ダミー」テーブルから選択するだけで実行できます。

図 29： SAOLEDB.11 を使用したリンク・サーバのテスト | コンテキスト： SQL Server

```
1  SELECT * FROM mem..sys.dummy
2  GO
```

正しく設定されていれば、次のような結果になります。

```
1> SELECT * FROM mem..sys.dummy
2> GO
dummy_col
-----------
        0
(1 rows affected)
```

代わりに次のメッセージが表示された場合、ターゲット・コンピュータで SQL Anywhere データベースを開始し忘れている可能性があります。

OLE DB provider “SAOLEDB.11” for linked server “mem” returned message “Database server not found”.

Msg 7399, Level 16, State 1, Server BRECK-PC¥TSUNAMI, Line 2

The OLE DB provider “SAOLEDB.11” for linked server “mem” reported an error. Auth entication failed.

Msg 7303, Level 16, State 1, Server BRECK-PC¥TSUNAMI, Line 2

Cannot initialize the data source object of OLE DB provider “SAOLEDB.11” for linked server “mem”.

(リンク・サーバ “mem” の OLE DB プロバイダ “SAOLEDB.11” がメッセージ「データベース・サーバが見つかりません」を返しました。メッセージ 7399、レベル 16、状態 1、サーバ BRECK-PC¥TSUNAMI、2 行目
リンク・サーバ “mem” の OLE DB プロバイダ “SAOLEDB.11” がエラー「認証に失敗しました」を報告しました。メッセージ 7303、レベル 16、状態 1、サーバ BRECK-PC¥TSUNAMI、2 行目
リンク・サーバ “mem” の OLE DB プロバイダ “SAOLEDB.11” のデータ・ソース・オブジェクトを初期化できません。)

図 30 は、SAOLEDB.11 を使用した手法 4 の INSERT です。これは、バックグラウンドで異なるプロバイダが使用されていることを除けば、図 20 の INSERT とまったく同じコードです。

図 30： SAOLEDB.11 を使用したリンク・サーバによるダウンロード | コンテキスト： SQL Server

```
1  INSERT  INTO  mem..dba.sa_target
2  SELECT  *
3    FROM mss_source
4  GO
```

SAOLEDB.11 の構成が完了したら、すぐに手法 5： OPENROWSET によるダウンロードでこのプロバイダを使用することができます。図 31 は、これまでの構成に問題がないかを確認するための単純なテストです。このテストでは、再びダミーから選択しますが、今回は粗削りの OPENROWSET 呼び出しを使用します。

図 31： SAOLEDB.11 を使用した OPENROWSET のテスト | コンテキスト： SQL Server

```
1  SELECT  *
2    FROM OPENROWSET  (  'SAOLEDB.11',
3    'sa_system_dsn'; 'dba';  'sql',
4    sys.dummy  )
5  GO
```

図 32 は、図 22 で示した OPENROWSET による挿入の SAOLEDB.11 バージョンです。

図 32： SAOLEDB.11 を使用した OPENROWSET によるダウンロード | コンテキスト： SQL Server

```
1  INSERT  INTO  OPENROWSET  (  'SAOLEDB.11',
2    'sa_system_dsn'; 'dba';  'sql',
3    dba.sa_target  )
4  SELECT  *
5    FROM mss_source
6  GO
```

パフォーマンス

パフォーマンスには、コンピュータのパフォーマンスとユーザのパフォーマンスの 2 種類があります。図 33 および図 34 に、コンピュータのパフォーマンスを示します。

図 33： 1 秒あたりのダウンロード・ロー数

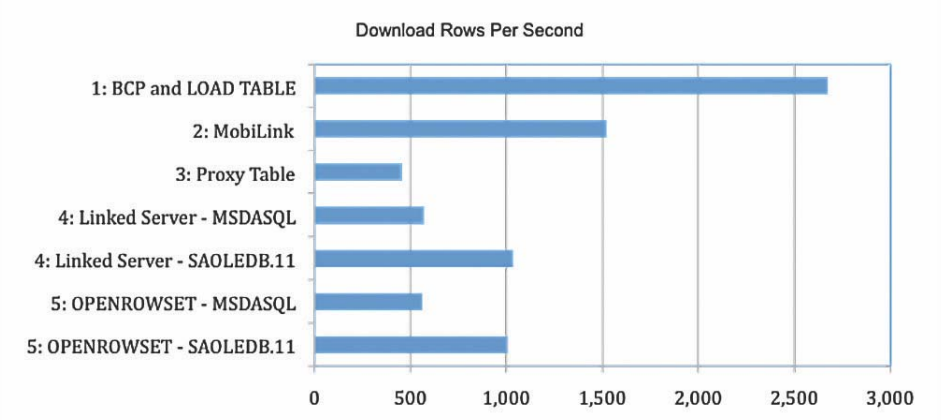


図 33 から、次のことが明らかです。

a) 2 つの手順からなる BCP および LOAD TABLE による手法

が飛びぬけて高速

b) プロキシ・テーブルによる手法が最も低速

c) Mobile Link による手法は次点 (2 位)

d) リンク・サーバによる手法と OPENROWSET による手法のパフォーマンスはほぼ同じだが、基礎となっている技術が同じため当然

e) SAOLEDB.11 プロバイダは MSDASQL よりはるかに勝る

f) プロキシ・テーブルによる手法は、MSDASQL を使用した場合と比べてそれほど劣らない

正直なところ、これらの結果のいくつかは予測どおりでしたが、Mobile Link は例外で、これほど高速なのは

ダウンロード手段	秒数 1	秒数 2	合計秒数	1 秒あたりのロー数	ロー・ロック数
手法 1： BCP および LOAD TABLE	446	274	720	2,674	0
手法 2： Mobile Link	533	733	1,266	1,521	1,925,469
手法 3：プロキシ・テーブル	4,230	—	4,230	455	1,925,469
手法 4： リンク・サーバ – MSDASQL	3,376	—	3,376	570	1,925,469
手法 4： リンク・サーバ – SAOLEDB.11	1,860	—	1,860	1,035	1,925,469
手法 5： OPENROWSET – MSDASQL	3,425	—	3,425	562	1,925,469
手法 5： OPENROWSET – SAOLEDB.11	1,916	—	1,916	1,005	1,925,469

図 34 の「ロー・ロック数」カラムから、LOAD TABLE が最も高速だった理由がわかります。すなわち、その他すべての手法では、SQL Anywhere が 190 万のロー・ロックを収集して解放しなければならなかったのに対し、LOAD TABLE は、パフォーマンスの観点から見て、本質的に非トランザクショナルです。また、SQL Server 側での BCP のパフォーマンスが優れていたこともあり、1 秒あたりのロー数が成功の主な (または唯一の) 評価基準である場合は、他の手法が差を縮めることさえないでしょう。

図 34 の「秒数 1」および「秒数 2」カラムは、最初の 2 つの手法にのみ適用されます。手法 1 については、BCP および LOAD TABLE についての時間がそれぞれ示されています。また、手法 2 については、Mobile Link ダウンロードの 2 つの段階の時間を示し、秒数 1 は Mobile Link サーバが SQL Server からローを選択 (SELECT)してダウンロード・ストリームに送信する時間 (190 万ローのイメージすべて)、秒数 2 は Mobile Link クライアントがローを SQL Anywhere データベースに挿入 (INSERT) する時間です。

人間のパフォーマンスの比較はこれほど単純ではありません。主観的な意見を次に挙げます。

a) BCP および LOAD TABLE による手法では、直接データベース接続ではなく外部ファイルが使用されるため、実装が難しくなる可能性があります。WAN、セキュリティ、暗号化などについて考えてみてください。

b) SQL Server 以外のターゲットやソースを扱う場合、BCP は完全に柔軟であるとはいえません。特に、データ自体に含まれる文字のあらゆる組み合わせで適切に機能する (また競合しない) BCP のローおよびカラム区切り文字列を指定するとなると、容易ではありません。 幸いにも、BCP 側で問題なく機能するものであれば、
LOAD TABLE の構文にはそれを処理する十分な柔軟性があります。

c) Mobile Link は圧倒的に柔軟性に優れた手法です。ソースとターゲット間のスキーマの大きな違いなど、「興味深い」データ変換が必要な場合は、Mobile Link が最も確実な方法でしょう。また、Mobile Link ではエンドツーエンドの暗号化も提供されます。

d) プロキシ・テーブルは好みがはっきり分かれ、中間がありません。サーバ結合が複雑な上、この記事で示したようにデータ伝送が単純で直線的なことから、長年、パフォーマンスが大きな問題となってきました。

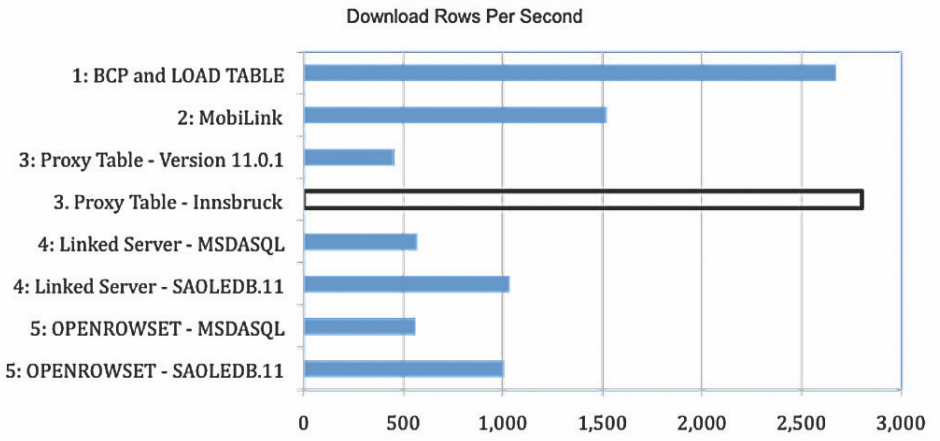
e) プロキシ・テーブルを好む人たちは、その柔軟性に惹かれています。SQL Anywhere プロキシ・テーブルと SQL Server のリンク・サーバおよび OPENROWSET による手法の両方を使用してきた人たちは、SQL Server 側についてさまざまな意見を持っているようです。問題は構文ではなく、十分に柔軟であるのに、ユーザに実際にどこか物足りないと感じさせかねないことです。

最新ニュース

時期バージョンの SQL Anywhere (コードネーム Innsbruck) は、この記事の執筆中にベータ・テスト段階に突入しました。ベータ版ソフトウェアの初期テストでは、プロキシ・テーブルの処理において、少なくとも直線的なデータ伝送に関しては、パフォーマンスが大幅に改善されていることが明らかになりました。

実際、Innsbruck を使用したプロキシ・テーブルによる手法は、この記事で紹介した他のどの手法よりも高速で、BCP および LOAD TABLE による手法さえも凌いでいます。図 18 に示したプロキシ・テーブルの INSERT - SELECT 文の実行時間は、SQL Anywhere 11.0.1 ソフトウェアを使用した場合は 4,230 秒であったのに対し、Innsbruck ではわずか 687 秒でした。図 35 を見れば、その効果は歴然です。

図 35： Innsbruck によるプロキシ・テーブルの改善



警告：図 35 には、Innsbruck を使用した他の手法の結果は含まれていないため、完全に公平な比較と言い切ることはできません。また、Innsbruck ソフトウェアのパフォーマンスは、一般公開前に変わる可能性があります。

とは言え、支持の幅を広げる過程でパフォーマンスが常に最大の障害物となってきたことを踏まえれば、プロキシ・テーブルの未来は明るいように思われます。

補足

ここで紹介してきた 5 つの手法が、SQL Server データベースと SQL Anywhere インメモリ・データベース間でデータをコピーする方法のすべてではありません。また、すべてのダウンロード手法をカバーしているわけでもなく、Sybase IQ や Oracle のような製品も取り上げていません。

この記事で扱っていないトピックの一部を次に挙げます。これは、ダウンロードおよびアップロードの両方について、SQL Server に関連するもののみです。

- SQL Anywhere の FROM OPENSTRING (FILE ...) 句を使用した、ロー・セットとしてのファイル内容の処理
- SQL Server の FROM OPENROWSET (BULK ...) 句を使用した、ロー・セットとしてのファイル内容の処理
- SQL Anywhere および SQL Server で多くの XML 対応機能を使用したデータのプッシュおよびプル
- SQL Anywhere の UNLOAD TABLE および UNLOAD SELECT を使用した、ロー・セットからのファイルの高速作成
- SQL Server の bcp.exe ユーティリティを使用した、ファイルからテーブルへのデータの高速アップロード
- SQL Server の BULK INSERT コマンドを使用した、ファイルからテーブルへのデータの高速アップロード
- Mobile Link を使用した SQL Server へのデータの直接アップロード、一般にアップロード・プロセスの駆動に使用されるトランザクション・ログがインメモリ SQL Anywhere データベースにはないことに対するスクリプト駆動型アップロード機能による対処
- プロキシ・テーブルを使用した SQL Server へのデータの直接アップロード
- リンク・サーバを使用した SQL Server へのデータの直接アップロード
- OPENROWSET を使用した SQL Server へのデータの直接アップロード

著者について

Breck Carter 氏は、SQL Anywhere データベース開発を専門とする独立系コンサルタントです。iAnywhere Solutions 顧客諮問委員会のメンバを務める一方、1993 年より Team Sybase に属し、最近 SQLA.StackExchange.com において SQL Anywhere Q&A の Web サイトを開設しました。同氏へのご連絡には、電子メール (Breck.Carter@gmail.com) をご利用ください。ブログは SQLAnywhere.blogspot.com で公開されています。

法的注意