チュートリアル：Ultra Light イベント通知の使用

SQL Anywhere 11 をはじめとして、Ultra Light データベースは、イベントおよび通知のサポートを提供します。データの修正、同期の要求およびコミットなどの特定のイベントが発生したときに、アプリケーションが通知を受け取ることができます。このチュートリアルでは、Windows データベース・コンソール・アプリケーションを使用してイベント通知の使用方法について説明します。

- 作成されるアプリケーションはマルチスレッド型であるため、新しい通知のチェックを行っても、アプリケーションとユーザとのやり取りが妨害されることはありません。データベースへの接続は、データベース・エンジン・クライアント・ライブラリを介して行われます。したがって uleng11.exe ユーティリティが使用されるため、さまざまなアプリケーションからデータベースに接続できます。具体的には、このチュートリアルでは管理プログラムの Interactive SQL を使用します。
- アプリケーションは、"Commit" システム・イベントの通知を取得するよう登録されています。データベースの修正時には、ユーザには更新されたデータがアプリケーション内に自動的に表示されます。アプリケーション・レベルで通知をチェックするので、チェックのたびにデータベースが余計な作業を実行しないで済みます。

このチュートリアルでは、単純化するためにエラー・チェックを最小限にしました。推奨案では、開発者はすべての戻り値をチェックすべきであるとされています。
サンプルは Visual C++ で記述されていますが、イベントおよび通知の API は、サポートされるすべての言語で提供されています。
プログラミング言語（エラー処理など）の相違や関数名の相違がありますが、提供されているサンプルは、C# や Visual Basic など他の言語でも使用できるように簡単に記述し直すことができます。

この特性の詳細については、オンライン・マニュアル内の「イベント通知の操作」を参照してください。

要件

このチュートリアルの作成には、以下が使用されています。

- SQL Anywhere 11.0.0 以上
- Visual Studio 2008 SP1、Microsoft Visual C++ 2008 - Ultra Light イベント通知は、Ultra Light がサポートされるすべてのプラットフォームで使用できます。

サンプルの実行

このチュートリアル用のすべてのファイルを格納するためのディレクトリを作成してください。このディレクトリを C:¥tutorial¥cpp であると仮定してチュートリアルを進めます。別の名前でディレクトリを作成する場合、これ以降はディレクトリ名をその名前に置き換えてください。

データベースのセットアップ：

- コマンド・プロンプトを開き、**C:¥tutorial¥cpp** に移動します。
- コマンド **ulcreate temp.udb** を使用して、Ultra Light データベースを作成します。

- Interactive SQL を使用して、データベースに接続します（**dbisql -ul -c dbf=C:¥tutorial¥cpp¥temp.udb**）。
- Interative SQL で、「付録 A - SQL コード」のコードをコピー・アンド・ペーストして実行します。この SQL スクリプトにより、顧客名のテーブルが作成されます。
- Interactive SQL を終了します。

アプリケーションのセットアップ:

「付録 B - C++ コード」に完全なコードがリストされています。

- お好きなコード・エディタを使用して、**"UltraLite Event Notifications.cpp"** ファイルを新規作成します。
- ヘッダを含め、Ultra Light ネームスペースを使用し、グローバル変数を宣言します。

```
#include <stdio.h>
#include "windows.h"
#include "uliface.h"
#include <stdlib.h>

using namespace UltraLite;

static char* const conn_string = "DBF=temp.udb;UID=dba;PWD=sql";
int flag=1;//flag to signal shutdown of watchthread.
```

- データベースのデータを表示する出力関数を作成します。

```
void output(ResultSet *rs)
{
	SYSTEMTIME lt;
	char *dest = (char*)malloc(sizeof(char)*30);

	//Just some output formatting.
	system("cls");
	printf(" _____ ¥n");
	printf("| %10s | %30s |¥n","cust_id","cust_name");
	printf("|____________|________________________________|¥n");
	while(rs->Next())
	{
		rs->Get(2).GetString(dest,30);
		printf("| %10d | %30s |¥n",(int)rs->Get(1),dest);
	}
	printf("|____________|________________________________|¥n");

	//Display the time for the last refresh of the data.
	GetLocalTime(&lt);
```

```
	printf("Last refresh done
at: %02d:%02d:%02d¥n",lt.wHour,lt.wMinute,lt.wSecond);

	//Clean up.
	free(dest);
	rs->Release();
}
```

- **通知をチェックする関数を作成します。**

db_check() **関数は、このアプリケーションのセカンド・スレッドのエントリ・ポイントです。この関数は、データベースから新しい通知が送信されているかどうかを（バックグラウンドで）モニタし、送信されていた場合は表示データをリフレッシュします。その間、アプリケーションの残りの部分は、ユーザの入力をチェックできます。**

```
DWORD WINAPI db_check(LPVOID lpParam)
{
	char* ulevent = (char*)malloc(sizeof(char)*20);
	//init objects and open connection
	ULSqlca _sqlca;
	_sqlca.Initialize();
	DatabaseManager* dbMgr = ULInitDatabaseManager(_sqlca);
	Connection* _conn = dbMgr->OpenConnection(_sqlca,conn_string);

	//test the connection
	if(_conn != NULL)
	{
		//this is it...
		_conn->RegisterForEvent("Commit","ULCustomer",NULL,true);

		//Do a first output.
		PreparedStatement *stmt = _conn->PrepareStatement(UL_TEXT("SELECT * from ULCustomer"));
		output(stmt->ExecuteQuery());
		do
		{
			//Check for any notifications on the default queue. If the table has been modified, refresh it.
			//There is only one regiestered event, so there is no need to check for any additional paramteres.
			_conn->GetNotification(NULL,100).GetString(ulevent,20);
			if(strcmp("Commit",ulevent)==0)
			{
				PreparedStatement *stmt = _conn->PrepareStatement(UL_TEXT("SELECT * from
ULCustomer"));
				output(stmt->ExecuteQuery());
			}
		}
```

```
            while(flag);
            _conn->Release();
        }
        else
        {
            printf("Error opening connection to database with the following connection string: %s¥n",conn_string);
        }
        free(ulevent);
        dbMgr->Shutdown(_sqlca);
        _sqlca.Finalize();
        return 0;
}
```

- **アプリケーションのエントリ・ポイントとなる** main **関数を作成します。**

起動時に、新しい通知をモニタする新たなスレッドが作成されます。その後は、単純化するために、main **関数はアプリケーションを終了するためのユーザ入力を待ちます。**

```
int main()
{
    DWORD dwThreadID;
    HANDLE watchthread =
CreateThread(NULL,0,db_check,NULL,0,&dwThreadID);

    getchar();
    flag=0;
    switch(WaitForSingleObject(watchthread,INFINITE))
    {
        case WAIT_OBJECT_0:
        {
            printf("All threads ended, cleaning up for application exit...¥n");
            break;
        }
        default:
        {
            printf("Wait error (%d)¥n", GetLastError());
            break;
        }
    }
    return 0;
}
```

アプリケーションのコンパイル:

- C:¥tutorial¥cpp フォルダに、アプリケーションをコンパイルするための “makefile” ファイルを新規作成します。「付録 C - Makefile コンテンツ」のコードをコピー・アンド・ペーストします。
- Visual Studio 2008 Command Prompt を開きます。C:¥tutorial¥cpp に移動し、nmake.exe を実行します。実行プログラム UltraLite Event Notifications.exe が作成されます。

サンプルの実行:

- ファイル UltraLite Event Notifications.exe を実行します。アプリケーションが起動されます。アプリケーションはデータベースのレコードを出力し、ユーザの入力を待ちます。
- Interactive SQL を使用してデータベースに接続します。コマンド・プロンプトをもう 1 つ開き、dbisql -ul -c dbf=C:¥tutorial¥cpp¥temp.udb を実行します。
- Interactive SQL で、以下の文を実行します。

```
INSERT INTO ULCustomer (cust_id, cust_name) VALUES (1, 'Test Customer');
COMMIT;
```

- Visual Studio Command Prompt で、“Last refresh time” が COMMIT 文の実行後の時間に更新されていて、新しいローが表示されていることに注意してください。任意のキーをタイプしてから、[Enter] を押してアプリケーションを終了します。
- Interactive SQL とコマンド・プロンプトを終了します。これでチュートリアルは終了です。

結論

Ultra Light イベント通知を使用して、データベースのコミット時にアプリケーションによって表示されるデータをリフレッシュすることができました。この処理は、すべてバックグラウンドで自動的に実行され、ユーザとアプリケーションのやり取りが妨害されることはありません。

Ultra Light イベント通知は、開発者に以下の重要な機能を提供します。

- 変更のプーリングがアプリケーション・レベルで行われるため、データベースで実行する必要がある作業の量を削減できます。
- 提供されるシステム・イベント (Commit、SyncComplete、および TableModified) に加えて、開発者が独自のイベントを宣言し、登録してトリガできます。

通知の取得は、新しいイベントに簡単に移植できる一般的な関数を使用して実行されます。したがって、アプリケーションでは、通常であればプーリング・メカニズムの修正が必要なデータベースの変更を行う必要がありません。また、データベースについても、プーリング・メカニズムを実装するためにスキーマを変更する必要がありません。

付録 A - SQL コード

```
create table ULCustomer (
cust_id integer not null primary key,
cust_name varchar(30)
);
```

```
create unique index ULCustomerName on ULCustomer (cust_name);
INSERT INTO ULCustomer (cust_id, cust_name) VALUES ( 2000, 'Apple St. Builders' );
INSERT INTO ULCustomer (cust_id, cust_name) VALUES ( 2001, 'Art''s Renovations' );
INSERT INTO ULCustomer (cust_id, cust_name) VALUES ( 2002, 'Awnings R Us' );
INSERT INTO ULCustomer (cust_id, cust_name) VALUES ( 2003, 'Al''s Interior Design' );
INSERT INTO ULCustomer (cust_id, cust_name) VALUES ( 2004, 'Alpha Hardware' );
INSERT INTO ULCustomer (cust_id, cust_name) VALUES ( 2005, 'Ace Properties' );
INSERT INTO ULCustomer (cust_id, cust_name) VALUES ( 2006, 'A1 Contracting' );
INSERT INTO ULCustomer (cust_id, cust_name) VALUES ( 2007, 'Archibald Inc.' );
INSERT INTO ULCustomer (cust_id, cust_name) VALUES ( 2008, 'Acme Construction' );
INSERT INTO ULCustomer (cust_id, cust_name) VALUES ( 2009, 'ABCXYZ Inc.' );
INSERT INTO ULCustomer (cust_id, cust_name) VALUES ( 2010, 'Buy It Co.' );
INSERT INTO ULCustomer (cust_id, cust_name) VALUES ( 2011, 'Bill''s Cages' );
INSERT INTO ULCustomer (cust_id, cust_name) VALUES ( 2012, 'Build-It Co.' );
INSERT INTO ULCustomer (cust_id, cust_name) VALUES ( 2013, 'Bass Interiors' );
INSERT INTO ULCustomer (cust_id, cust_name) VALUES ( 2014, 'Burger Franchise' );
INSERT INTO ULCustomer (cust_id, cust_name) VALUES ( 2015, 'Big City Builders' );
INSERT INTO ULCustomer (cust_id, cust_name) VALUES ( 2016, 'Bob''s Renovations' );
INSERT INTO ULCustomer (cust_id, cust_name) VALUES ( 2017, 'Basements R Us' );
INSERT INTO ULCustomer (cust_id, cust_name) VALUES ( 2018, 'BB Interior Design' );
INSERT INTO ULCustomer (cust_id, cust_name) VALUES ( 2019, 'Bond Hardware' );
INSERT INTO ULCustomer (cust_id, cust_name) VALUES ( 2020, 'Cat Properties' );
INSERT INTO ULCustomer (cust_id, cust_name) VALUES ( 2021, 'C & C Contracting' );
INSERT INTO ULCustomer (cust_id, cust_name) VALUES ( 2022, 'Classy Inc.' );
INSERT INTO ULCustomer (cust_id, cust_name) VALUES ( 2023, 'Cooper Construction' );
INSERT INTO ULCustomer (cust_id, cust_name) VALUES ( 2024, 'City Schools' );
INSERT INTO ULCustomer (cust_id, cust_name) VALUES ( 2025, 'Can Do It Co.' );
INSERT INTO ULCustomer (cust_id, cust_name) VALUES ( 2026, 'City Corrections' );
INSERT INTO ULCustomer (cust_id, cust_name) VALUES ( 2027, 'City Sports Arenas' );
INSERT INTO ULCustomer (cust_id, cust_name) VALUES ( 2028, 'Cantelope Interiors' );
INSERT INTO ULCustomer (cust_id, cust_name) VALUES ( 2029, 'Chicken Franchise' );
```

付録 B - C++ コード

```
#include <stdio.h>
#include "windows.h"
#include "uliface.h"
#include <stdlib.h>

using namespace UltraLite;

static char* const conn_string = "DBF=temp.udb;UID=dba;PWD=sql";
int flag=1;//flag to signal shutdown of watchthread.

void output(ResultSet *rs)
```

```
{
	SYSTEMTIME lt;
	char *dest = (char*)malloc(sizeof(char)*30);

	//Just some output formatting.
	system("cls");
	printf(" _____ ¥n");
	printf("| %10s | %30s |¥n","cust_id","cust_name");
	printf("|____________|_____________________________|¥n");
	while(rs->Next())
	{
		rs->Get(2).GetString(dest,30);
		printf("| %10d | %30s |¥n",(int)rs->Get(1),dest);
	}
	printf("|____________|_____________________________|¥n");

	//Display the time for the last refresh of the data.
	GetLocalTime(&lt);
	printf("Last refresh done at: %02d:%02d:%02d¥n",lt.wHour,lt.wMinute,lt.wSecond);

	//Clean up.
	free(dest);
	rs->Release();
}

DWORD WINAPI db_check(LPVOID lpParam)
{
	char* ulevent = (char*)malloc(sizeof(char)*20);
	//init objects and open connection
	ULSqlca _sqlca;
	_sqlca.Initialize();
	DatabaseManager* dbMgr = ULInitDatabaseManager(_sqlca);
	Connection* _conn = dbMgr->OpenConnection(_sqlca,conn_string);

	//test the connection
	if(_conn != NULL)
	{
		//this is it...
		_conn->RegisterForEvent("Commit","ULCustomer",NULL,true);

		//Do a first output.
		PreparedStatement *stmt = _conn->PrepareStatement(UL_TEXT("SELECT * from ULCustomer"));
		output(stmt->ExecuteQuery());
		do
		{
```

```
                //Check for any notifications on the default queue. If the table has been modified, refresh it.
                //There is only one regiestered event, so there is no need to check for any additional paramteres.
                _conn->GetNotification(NULL,100).GetString(ulevent,20);
                if(strcmp("Commit",ulevent)==0)
                {
                    PreparedStatement *stmt = _conn->PrepareStatement(UL_TEXT("SELECT * from
ULCustomer"));
                    output(stmt->ExecuteQuery());
                }
            }
            while(flag);
            _conn->Release();
        }
        else
        {
            printf("Error opening connection to database with the following connection string: %s¥n",conn_string);
        }
        free(ulevent);
        dbMgr->Shutdown(_sqlca);
        _sqlca.Finalize();
        return 0;
}

int main()
{
    DWORD dwThreadID;
    HANDLE watchthread = CreateThread(NULL,0,db_check,NULL,0,&dwThreadID);

    getchar();
    flag=0;
    switch(WaitForSingleObject(watchthread,INFINITE))
    {
        case WAIT_OBJECT_0:
        {
            printf("All threads ended, cleaning up for application exit...¥n");
            break;
        }
        default:
        {
            printf("Wait error (%d)¥n", GetLastError());
            break;
        }
    }
    return 0;
}
```

付録 C - Makefile コンテンツ

```
IncludeFolders=/I "$(WindowsSdkDir)¥Include" /I "$(SQLANY11)¥SDK¥Include"
LibraryFolders=/LIBPATH:"$(SQLANY11)¥UltraLite¥win32¥386¥Lib¥vs8"
/LIBPATH:"$(WindowsSdkDir)¥Lib"
Libraries=ulbase.lib ulimpc.lib
CompileOptions=/c /W3 /Zi /DWIN32 /DUL_USE_DLL
"UltraLite Event Notifications.exe" : "UltraLite Event Notifications.obj"
    link /NOLOGO /DEBUG "UltraLite Event Notifications.obj" $(LibraryFolders)
$(Libraries)
"UltraLite Event Notifications.obj" : "UltraLite Event Notifications.cpp"
    cl $(CompileOptions) $(IncludeFolders) "UltraLite Event Notifications.cpp"
```