

Re: Idea of SQL integration

Source:

<http://coding.derkeiler.com/Archive/Delphi/borland.public.delphi.non-technical/2006-07/msg01895.html>

- *From:* "William Egge" <wegge@xxxxxxxxxxxxxxxx>
 - *Date:* Sat, 15 Jul 2006 19:55:50 -0400
-

I have done something similar with ADO.

Here is my interface section:

```
unit ADOHelp;
{.$define LogSQL}

interface
uses
{$ifdef delphiado}
AdoInt,
{$else}
ADODB_TLB,
{$endif}
Fn_FileToString, Fn_GetModuleName, SysUtils, Fn_StringIndex, Windows,
MD5,
Log, Dialogs,
FastStrings,
Fn_AccmOfStrings
{$ifdef ver140}
, Variants
{$endif}
;

type
TTableType = (ttUnknown, ttTable, ttSystemTable);

TTableDef_v1 = packed record
TableName: string[100];
TableType: TTableType;
end;

TTableColumnDef_v1 = packed record
ColumnName: string[50];
OrdinalPosition: Integer;
Nullable: Boolean;
AdoDataType: Integer;
CharMaxLen: Integer;
```

Re: Idea of SQL integration

end;

```
TOfficeType = (otXCell, otAccess);
```

```
// Connection functions
```

```
function OpenConnection(const ConnectionString: string): Connection;  
function OpenConnectionFromFile(const FileName: string): Connection;  
function OpenConnectionLocalDirFile(const LocalFile: string): Connection;
```

```
// Office Connection string
```

```
function OfficeConnStr(const FileName: string; OfficeType: TOfficeType;  
XCellHeader: Boolean = True; UserID: string = ""; Password: string = ""):  
string;  
{  
Using Excell  
opening the worksheet sql  
select * from [Sheet1$]
```

Use a # sign for periods in ado field object column names.

Ex: if Column name is Per. Start then access ado field as Per# Start

if there are no column headers then ado field object column names are
F1, F2, F3...

```
select * on an empty spreadsheet only 1 column, F1  
}
```

```
// MS Access
```

```
function OpenMSAccess(const FileName: string; UserID: string = ""; Password:  
string = "): Connection;  
function MSAccessConnStr(const FileName: string; UserID: string = ""; Password:  
string = "): string;  
function IsJet(const Conn: Connection): Boolean;
```

```
//SQLServer
```

```
function OpenMSSQL(const Server, Database: string): Connection;
```

```
// Recordset functions
```

```
function FireHose(const Conn: Connection; const SQL: string; CacheSize: Integer  
= 100): Recordset;  
function ClientDisconnectedRec(const Conn: Connection; const SQL: string):  
Recordset;  
function EditableRecordset(const Conn: Connection; const SQL: string; Batch:  
Boolean = False; ReturnNoRecs: Boolean = False): Recordset;  
function ExecCMD(const Conn: Connection; const CmdText: string): Integer;  
function InsertSQLSvr(const Conn: Connection; const InsertSQL: string): Integer;
```

```
// Transaction across many
```

```
procedure BeginAllTrans(const ConnList: array of Connection);  
procedure RollbackAllTrans(const ConnList: array of Connection);  
procedure CommitAllTrans(const ConnList: array of Connection);
```

Re: Idea of SQL integration

```
// Schema functions
function GetTables(const Conn: Connection): Recordset;
function GetTableType(const TableSchema: Recordset): TTableDef_v1;
function GetTableColumns(const Conn: Connection; const TableName: string):
Recordset;
function GetColumnDef(const Column: Recordset): TTableColumnDef_v1;

// Variant routines
function VarToInt(V: OleVariant): Integer;
function VarToExtended(V: OleVariant): Extended;

// Custom Recordset Data Holder
type
TRecFieldDef = record
FieldName: string;
AdoType: LongWord;
Size: Integer;
end;
TRecFieldDefArray = array of TRecFieldDef;

function MakeFieldDefArray(Fields: array of TRecFieldDef): TRecFieldDefArray;

function CustomRecordset(Fields: array of TRecFieldDef): Recordset; overload;
function CustomRecordset(const CopyStructureFrom: Recordset): Recordset;
overload;
procedure AddData(const Source, Dest: Recordset; CurrentRowOnly: Boolean =
False);
function RecField(const FieldName: string; AdoType: LongWord; Size: Integer
= -1): TRecFieldDef;
function IntRecField(const FieldName: string): TRecFieldDef;
function CharRecField(const FieldName: string; Length: Integer): TRecFieldDef;
function StringRecField(const FieldName: string): TRecFieldDef;
function DateRecField(const FieldName: string): TRecFieldDef;
function BoolRecField(const FieldName: string): TRecFieldDef;
function CurrencyRecField(const FieldName: string): TRecFieldDef;

function BuildRecordset(Fields: array of TRecFieldDef; Values: OleVariant):
Recordset;

function RecordMD5(const R: Recordset; const TableName: string; out TextData:
string): TDigestStr;
function RecordText(const R: Recordset; const TableName: string): string;

implementation
```

"Ralf Mimoun" <nospam@xxxxxxxxxx> wrote in message

Re: Idea of SQL integration

news:44b94334\$1@xxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

Frenk wrote:

After almost full-time DB programming, I was thinking about how to integrate RDBMS & Delphi. I am wondering why isn't possible to solve this issues like compiler directive e.g. ASM. We all know that there is no general solution for multiple DB vendors (sas for multiple CPU-s too). Isn't just a great idea like this:

I don't know. It will be problematic to adopt it to alle the db systems flying around.

I use a very simple approach. My central datamodule has a function called NewQuery. It gets a SQL statement and another parameter (optional): Open, Execute, ExecuteAndFree. The function creates the query object (mostly a TDBISAMQueryExt), sets session, database etc, adds the SQL command and does what the parameter say. Easy enough to use for me, it's everywhere in my application. And yes, that's a place where I use with:

```
with MainDataModule.NewQuery('SELECT COUNT(*) AS RecCount FROM Customers',  
nqOpen) do begin  
try  
Result := FieldByName('RecCount').AsInteger;  
finally  
Free;  
end;  
end;
```

Ralf