



Akademia BSC



FireDAC

FireDAC dla serwerów SQL

Akademia BSC

BSC Polska © 2018

Szkolenie

Sekcja



FireDAC

Wprowadzenie do FireDAC

Pierwsze kroki

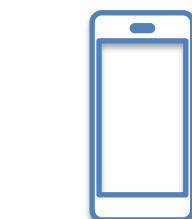
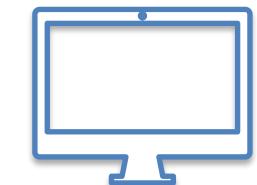
Definicja połączenia

System opcji FireDAC

Dokumentacja i przykłady



FireDAC



Pierwsze kroki

Połączenie z bazą danych

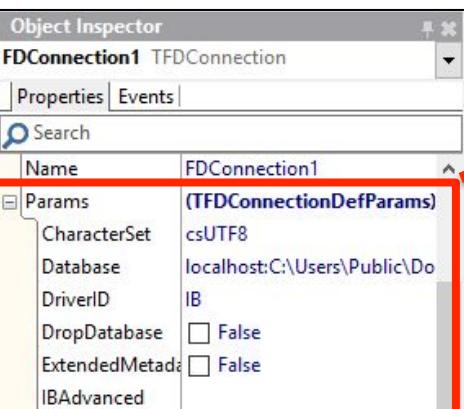
TFDConnection

– Metoda Open

- *Parametry połączenia przekazane w formie ConnectionString'a (oddzielone separatorem)*

– Object Inspector

- *Double-click FDConnection*
- *Własność Params*



```
FDConnection1.Open('DriverID=SQLite;'+
'Database=C:\Users\Public\' +
'Documents\Embarcadero\Studio\19.0\' +
'Samples\data\FDDemo.sdb;');
```

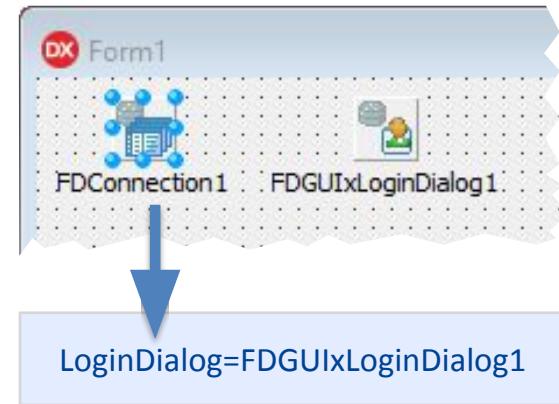
```
CustNo := FDConnection1.ExecSQLScalar(
'select count(*) from Customers');
```

```
Button1.Caption := CustNo;
```

```
FDConnection1.Open('DriverID=IB;'+
'Database=localhost:C:\Users\Public\' +
'Documents\Embarcadero\Studio\19.0\' +
'Samples\data\fddemo.ib;'+
'User_Name=SYSDBA;'+
'Password=masterkey;'+
'CharacterSet=UTF8;');
```

Login dialog

```
with FDGUIxLoginDialog1.VisibleItems do begin
  Clear;
  Add('Database=Baza');
  Add('User_name=Użytkownik');
  Add('Password=Hasło');
end;
FDConnection1.LoginDialog := FDGUIxLoginDialog1;
FDConnection1.Connected := True;
```



Dodatkowy komponent

- Możliwość dostosowania
 - Wyświetlane dodatkowych pól
 - Tłumaczenie
- Ponawianie logowania
- Obsługa żądania od serwera zmiany hasła
- Zapamiętanie historii logowania
 - Zapisywana z hasłem (otwartym tekstem)

FireDAC Units

FireDAC.Comp	Zawiera komponenty FireDAC
FireDAC.Moni	Monitorowanie
FireDAC.Phys	Sterowniki FireDAC i funkcje specyficzne dla wybranych serwerów
FireDAC.Stan	Warstwa standard FireDAC

```
// ***** TFDConnection
FireDAC.Stan.Intf,
FireDAC.Stan.Option,
FireDAC.Stan.Error,
FireDAC.UI.Intf,
FireDAC.Phys.Intf,
FireDAC.Stan.Def,
FireDAC.Stan.Pool,
FireDAC.Stan.Async,
FireDAC.Phys,
FireDAC.Comp.Client,
// ***** wymagane
FireDAC.VCLUI.Wait,
Data.DB,
// ***** SQLite driver
FireDAC.Stan.ExprFuncs,
FireDAC.Phys.SQLiteDef,
FireDAC.Phys.SQLite;
```

Wybrane metody TFDConnection

Wykonanie SQLa

- ExecSQLScalar
- ExecSQL

Metadane

- GetTableNames
- GetFieldNames
- GetIndexNames
- GetStoredProcNames

Sekwencje / Generatory

- GetLastAutoGenValue

Transakcje

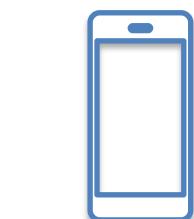
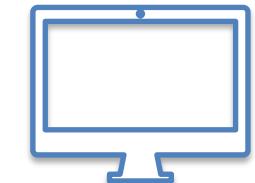
- StartTransaction
- Commit
- CommitRetaining
- Rollback

Inne (wybrane)

- Ping
- Online
- Offline
- GetInfoReport
- CloneConnection



FireDAC



Definicja połączenia

Definicja połączenia

Skrót / Rekord

- Zawiera pełen zbiór parametrów połączeniowych
- Zawiera opcje połączenia i zbiorów danych
- Może być nazwany
- Może być utrwalony w bazie definicji połączeń

```
params := TStringList.Create;
params.Add('Server=127.0.0.1');
params.Add('Database=addemo');
FDManager.AddConnectionDef('myconn',
  'MySQL', params, False);
// False = Persistent
```

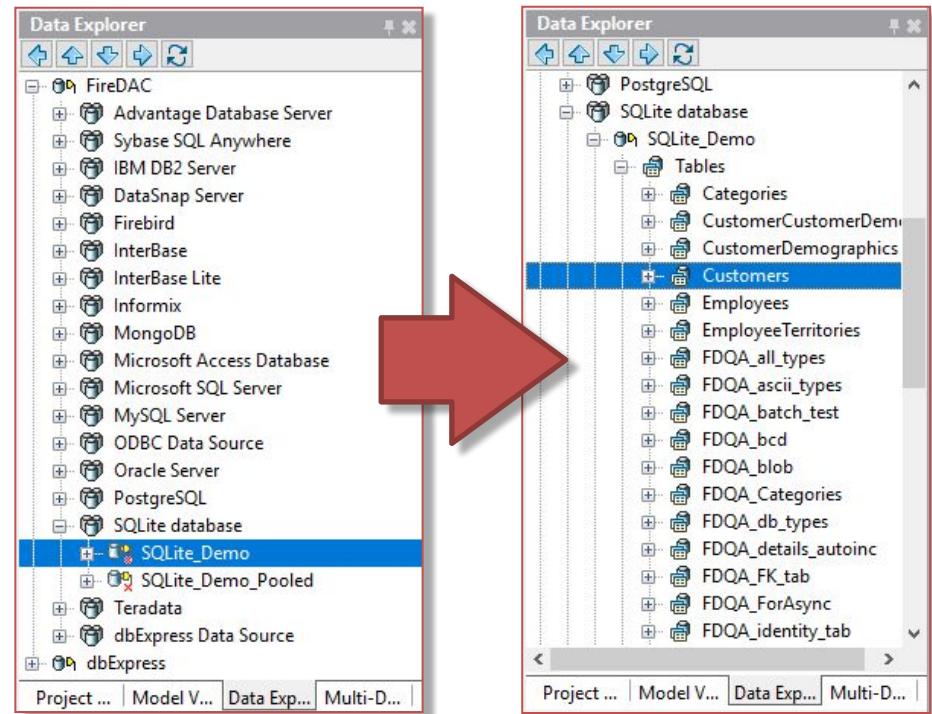
Geneza

- BDE Administrator

Jak zdefiniować nowe połączenie

Definiowanie

- Data Explorer
 - Wbudowany w IDE
- FDExplorer
 - FireDAC Explorer
 - Menu Tools
- W bazie definicji
 - Plik INI
- W kodzie



Nowa definicja - wersja pełna

Kod dodający nową definicję połączenia

- Wersja pełna
- Obiekt z parametrami zgodny z parametrami oczekiwanyimi przez wybrany serwer SQL

```
oDef := FDManager.ConnectionDefs.AddConnectionDef;  
oDef.Name := cNameConnDef;  
oParams := TFDPhysMSSQLConnectionDefParams(  
  oDef.Params);  
oParams.DriverID := 'MSSQL';  
oParams.Database := 'Northwind';  
oParams.UserName := 'sa';  
oParams.Password := '.....';  
oParams.Server := '127.0.0.1';  
oParams.OSAuthent := false;  
oParams.MARS := false;  
oDef.MarkPersistent;  
oDef.Apply;
```

```
FDManager.ConnectionDefFileName := ExtractFilePath(Application.ExeName) +  
'myconndef.ini';  
FDManager.ConnectionDefFileAutoLoad := True;  
oConn := TFDConnection.Create(nil);  
oConn.ConnectionDefName := 'myconn';  
oConn.Connected := True;
```

FireDAC.Comp.Client.TFDCustomManager.ConnectionDefFileName

Rodzaje definicji połączeń

Typ	Opis	Za	Przeciw
Utrwalona	Definicja ma unikalną nazwę. Ustawiania w FDManager i jest zapisana w pliku INI	Definiowana raz i może być wielokrotnie używana przez wiele aplikacji. Można ustalić pulę połączeń	Parametry są widoczne publicznie. Aby zobaczyć nowo dodaną definicję trzeba ponownie uruchomić manager
Prywatna	Ma unikalną nazwę i jest ustawiana w FDManager ale nie zostaje zapisana w pliku INI	Parametry nie są widoczne poza aplikacją Można ustalić pulę połączeń	Nie może być używana przez inne programy Nie może być zdefiniowania w projektancie.
Tymczasowa	Nie ma nazwy, nie jest zapisana w INI, nie można zarządzać nią w FDManager	Podobnie jak w wielu innych komponentach pakietu DAC. W kodzie / komponencie są zawarte wszystkie informacje o połączeniu.	Jak wyżej. Nie można utworzyć puli połączeń

Użycie definicji połączenia

The screenshot illustrates the process of using connection definitions in the Delphi IDE to bind data to a form.

Data Explorer: A window showing the database schema of the `SQLite_Demo` database, specifically the `Customers` table.

Object Inspector: A window showing the properties of the `Sqlite_demoConnection` component. The `ConnectionDef` property is set to `SQLite_Demo`, and the `DriverName` property is set to `SQLite`.

Form1: A Windows application form containing a `TDataSource` component named `Sqlite_demoConnection` and a `TTable` component named `CustomersTable`. The `Sqlite_demoConnection` component is highlighted with a red arrow pointing from the Object Inspector.

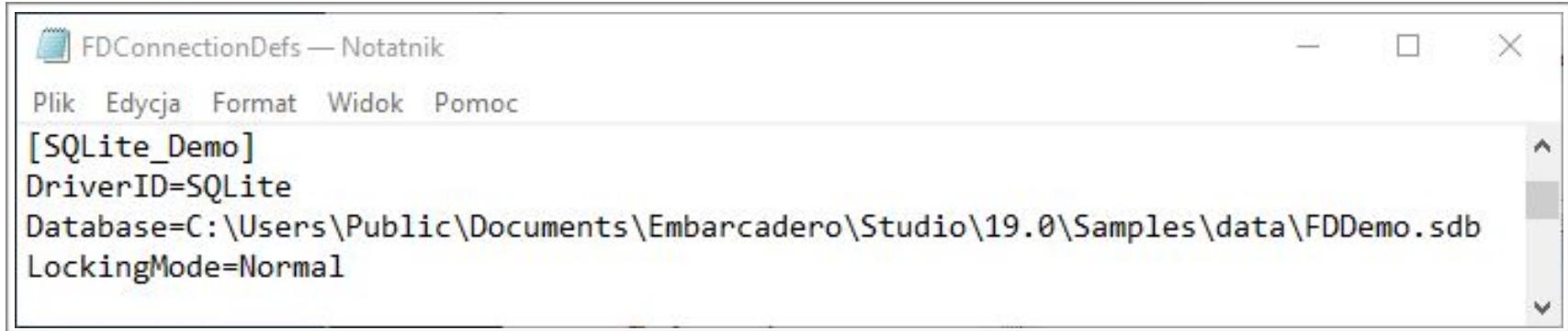
Form1 Data View: A grid view showing the data from the `Customers` table. The data includes columns: CustomerID, CompanyName, Contact, and ContactTitle.

Annotations:

- A blue arrow labeled "drag & drop" points from the `Customers` table in the Data Explorer to the `Sqlite_demoConnection` component in the Form1 designer.
- An orange arrow points from the `Sqlite_demoConnection` component in the Form1 designer to the `Sqlite_demoConnection` component in the Object Inspector.

CustomerID	CompanyName	Contact	ContactTitle
ALFKI	Alfreds Futterkiste	Maria Anders	Sales Representative
ANATR	Ana Trujillo Emparedados y helados	Ana Trujillo	Owner
ANTON	Andrea Moreno Taqueria	Antonio Moreno	Owner
AROSA	Arabella S立て snabbkop	Thomas Mandell	Marketing Manager
BERGS	Iceland Delikatessen	Christina Berglund	Owner
BLAUS	Blau delikatessen	Hanna Mel江	Owner
BLONP	CustomeTable	Frederick Blom	Owner
BOLID	Bolido Comidas preparadas	Martin Svensson	Owner
BONAP	Bon app'	Laurence Berman	Owner
BOTTM	Bottom-Dollar Markets	Elizabeth Harris	Owner
BSBEV	B's Beverages	Victoria Chang	Owner
CACTU	Cactus Comidas para llevar	Patricia Mikkelsen	Owner

Baza definicji połączeń

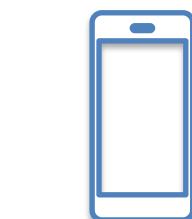
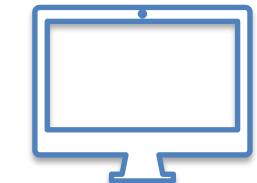


```
[SQLite_Demo]
DriverID=SQLite
Database=C:\Users\Public\Documents\Embarcadero\Studio\19.0\Samples\data\FDDemo.sdb
LockingMode=Normal
```

Domyślna lokalizacja	C:\Users\Public\Documents\Embarcadero\Studio\FireDAC\
Plik z bazą definicji	FDConnectionDefs.ini
Wyszukiwanie pliku	<ol style="list-style-type: none">1) Lokalny folder aplikacji (środowiska)2) Jeden z folderów systemowych3) FireDAC Home (domyślna lokalizacja)



FireDAC



System opcji FireDAC

Opcje FireDAC

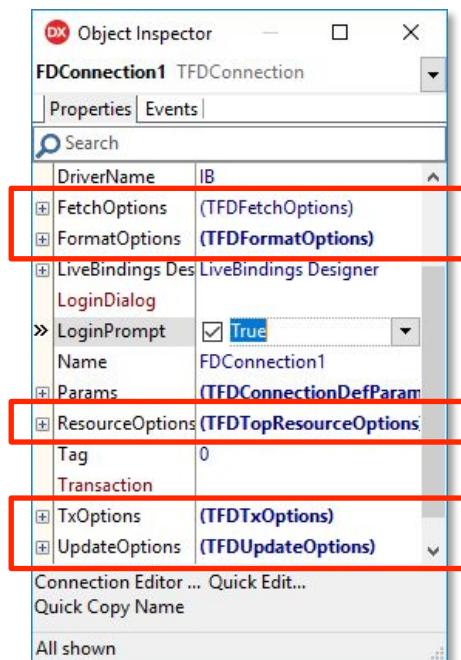
Poziomy opcji

- Kaskada
 - *Dziedziczone*
 - *Opcje mogą zostać nadpisane na niższym poziomie*



Kategorie opcji

- Fetch - pobierania danych
- Format - wyświetlanie danych
- Update - aktualizacji danych
- Resource - używania zasobów
- Tx - działania transakcji



FireDAC system opcji

Przykłady

1. Konwersja typów danych w ramach połączenia

Konwersja TFmtBCDField

NUMBER(19:4) na pola TCurrencyField

2. Ustawienie globalnych opcji pobierania danych dla całej aplikacji

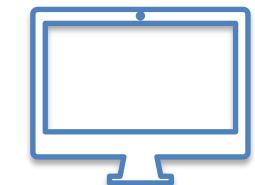
Wyłączenie natychmiastowego pobierania BLOBów, detali i metadanych, wyłączenie cache'owania, pobieranie wierszy w paczkach po 1000

```
with FDConnection1.Options.FormatOptions
do begin
  OwnMapRules := True;
  MapRules.Clear;
  with MapRules.Add do begin
    PrecMax := 19;
    PrecMin := 4;
    SourceDataType := dtFmtBCD;
    TargetDataType := dtCurrency;
  end;
end;
```

```
with FDManager.FetchOptions do begin
  Items := [];
  Cache := [];
  RowsetSize := 1000;
end;
```



FireDAC



Dokumentacja i przykłady

- Dokumentacja online i offline
- Bazy przykładowe FireDAC Demo
- Przykłady

Dokumentacja

<http://docwiki.embarcadero.com/RADStudio/Tokyo/en/FireDAC>

Online: Docwiki Embarcadero

Offline: Wbudowany help

Kluczowe tematy

Overview	Przegląd tematów i katalog kluczowych informacji Szybkie odnalezienie tematu
Getting Started	Wprowadzenie bogato ilustrowane przykładami
Working with Connections	Łączenie z bazami danych, utrzymywanie połączenia, konfiguracja, transakcje, obsługa błędów
Working with Commands	Wykonywanie poleceń SQL, pobieranie danych i aktualizacja danych
Working with DataSets	Sortowanie, filtrowanie, wyszukiwanie, master-detail, pola wyliczane
Editing Data	Pola auto-numerowane, buforowanie danych,
Working with Metadata	Pobieranie meta-informacji o strukturze bazy

Przykładowe bazy danych

Opisane w dokumentacji

- Getting Started → Demo Databases

Gotowe bazy demo

- MS Access, SQLite, Interbase

Tworzenie bazy demo

1. Założenie nowej (pustej) bazy danych

- Korzystając ze standardowych narzędzi platformy DB

2. Dodanie definicji połączenia

- Definicja z parametrami tej bazy
- Domyślne nazwy definicji: DB2_Demo, FB_Demo, IB_Demo, MySQL_Demo, Oracle_Demo, PG_Demo, itp.

3. Uruchomienie pliku bat

- create****.bat : np.: createMSSQL.bat, createIB.bat, ...
- Tworzy strukturę i dodaje dane
- folder z przykładami FireDAC

FireDAC Samples

Przykłady FireDAC

http://docwiki.embarcadero.com/CodeExamples/Tokyo/en/Code_Examples_by_Functionality#F

C:\Users\Public\Documents\Embarcadero\Studio\19.0\Samples\Object Pascal\Database\FireDAC\

Sekcja



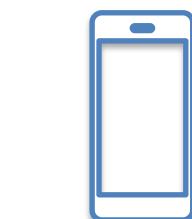
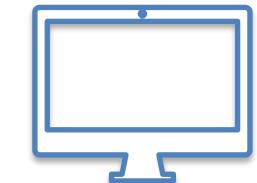
TFDConnection

FireDAC

- Polecenia SQL
- Obsługa błędów
- Preprocesor FireDAC
- Odzyskiwanie połączenia
- Diagnostyka połączenia
- Pobieranie metadanych



FireDAC



Polecenia SQL w TFDConnection

FDConnection ExecSQL

FDConnection.ExecSQL

ExecSQL (SQL)

ExecSQL (SQL, IsIgnoreNotExistObjects)

ExecSQL (SQL, Params)

ExecSQL (SQL, Params, ParamsDataType)

```
FDConnection1.ExecSQL('truncate table tab1');

FDConnection1.ExecSQL('drop table tab2', True);

FDConnection1.ExecSQL('delete from mytab where id > :p1', [1000]);

FDConnection1.ExecSQL('update mytab where id = :p1 set blobfld = :blob',
[1000, StringOfChar('x', 100000)], [ftInteger, ftBLOB]);
```

FDConnection ExecScalar

FDConnection.ExecScalar

```
ExecScalar (SQL) : Variant;  
ExecScalar (SQL, Params) : Variant;
```

```
sVersion := FDConnection1.ExecSQLScalar('select @@version');  
  
sName := FDConnection1.ExecSQLScalar('select name from {id Employees}  
where id = :id', [100]);  
  
sName2 := FDConnection1.ExecSQLScalar('select name from {id Employees}  
where id = :id', [100], [ftLargeInt]);
```

ExecSQL - przykład

Stworzenie tabeli i wypełnienie jej danymi

```
FDConnection1.ExecSQL('drop table if exists tab1');

FDConnection1.ExecSQL('create table tab1 (id int primary key not null,
name varchar(50))');

FDConnection1.StartTransaction;

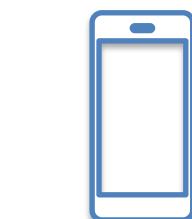
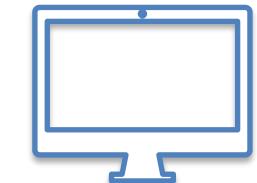
FDConnection1.ExecSQL('insert into tab1 values (:id,:name)', [1,'BP']);

FDConnection1.ExecSQL('insert into tab1 values (:id,:name)', [2,'BSC']);

FDConnection1.Commit;
```



FireDAC



Obsługa błędów

Uniwersalna obsługa błędów

Niezależna

- od platformy DB
- FireDAC zgłasza wyjątek
`EFDDDBEngineException`

Pola

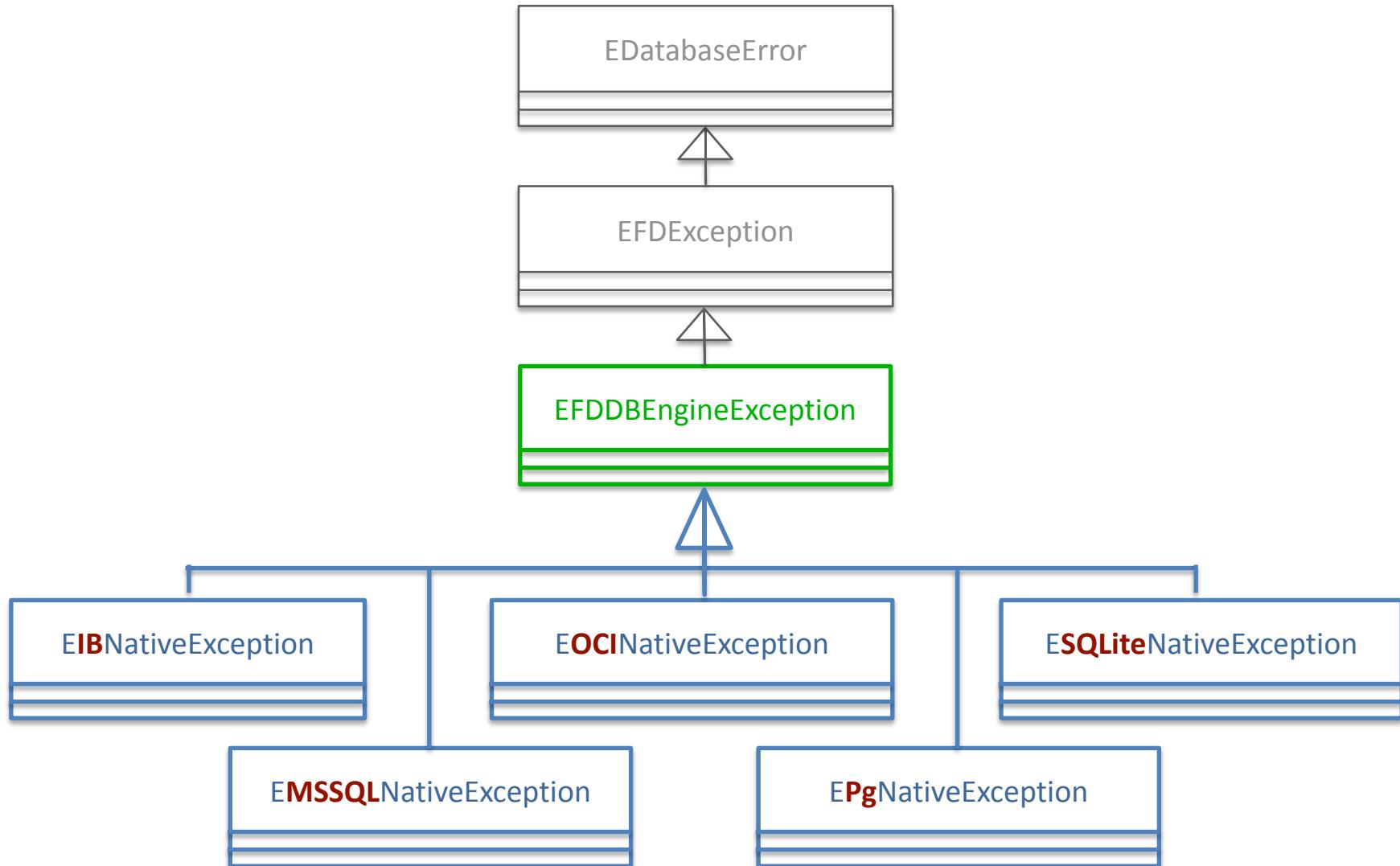
- `Kind`
 - *Typ wyliczeniowy*
 - *Informuje o najczęstszych błądach albo zwraca ekOther*
- `ErrorCode`
 - *Rzeczywisty nr błędu przesłany przez serwer SQL*

Właściwość	Opis
<code>Kind</code>	Rodzaj błędu (niezależny od platformy DB)
<code>Message</code>	Komunikat
<code>ErrorCode</code>	Kod błędu, wygenerowany przez platformę DB
<code>Errors</code>	Kolekcja błędów TFDBError
<code>ErrorCount</code>	Liczba błędów w kolekcji

Kind

`ekOther, ekNoDataFound, ekTooManyRows,`
`ekRecordLocked, ekUKViolated,`
`ekFKViolated, ekObjNotExists,`
`ekUserPwdInvalid, ekUserPwdExpired,`
`ekUserPwdWillExpire, ekCmdAborted,`
`ekServerGone, ekServerOutput,`
`ekArrExecMalfunc, ekInvalidParams`

Hierarchia dziedziczenia wyjątków FireDAC



Obsługa zgłaszanych błędów

```
try
  FDQuery1.ExecSQL(
    'insert into MyTab(code, name) values (:code, :name)', 
    [100, 'Tokyo']);
except
  on E: EFDBEngineException do begin
    if E.Kind = ekUKViolated then
      ShowMessage('Please enter unique value !');
    raise;
  end;
end;
```

```
FDConnection1.StartsTransaction;
try
  FDQuery1.ExecSQL;
  FDConnection1.Commit;
except
  on E: EFDBEngineException do begin
    FDConnection1.Rollback;
    // do something here
    raise;
  end;
end;
```

Metody przechwytywania błędów

Przychwytywanie wyjątków

- Sekcja try .. except

Zdarzenia

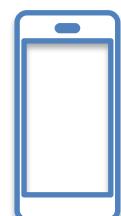
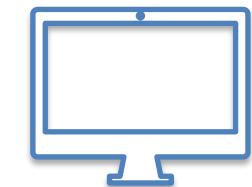
- TFDQuery.OnError
- TFDConnection.OnError
- TFDQuery.OnExecuteError
- TFDQuery.OnUpdateError
- TFDConnection.OnLost,
OnRestored, OnRecover

FDConnection.OnError

```
procedure TForm1.FDConnection1Error(ASender, AInitiator: TObject;
  var AException: Exception);
var
  Kind: TFDCommandExceptionKind;
begin
  if AException is EFDBEngineException then begin
    Kind := (AException as EFDBEngineException).Kind;
    if Kind = ekUserPwdInvalid then begin
      Self.Caption := 'Error! Invalid user/password.';
      FreeAndNil (AException);
      AException := EAbort.Create('silent exception');
    end
  end;
end;
```



FireDAC



Preprocesor FireDAC

- Działanie TFDQuery i proces komunikacji z DBMS
- Parametry polecenia
- Preprocesor FireDAC
- Procedury składowane

Preprocesor FireDAC

Polecenia preprocesora

- Wstawianie w tekście polecenia SQL
- Zostają przetworzone w czasie przygotowania polecenia
- Nie są wysyłane do serwera

```
SELECT {ucase(CompanyName)} FROM  
"Customers"
```

```
{IF Oracle} SELECT * FROM OracleTab {fi}  
{IF MSSQL} SELECT * FROM MSSQLTab {fi}
```

```
SELECT count(*) FROM {id Customers}
```

```
with FDQuery1 do begin  
  SQL.Text := 'SELECT * FROM &TableName';  
  MacroByName('TableName').AsRaw :=  
    'Orders';  
  Open;  
end;
```

Przykładowe funkcje znakowe

ASCII (*str*)
CHAR (*code*)
CHAR_LENGTH (*str*)
CONCAT (*str1, str2*)
INSERT (*str1, start, length, str2*)
LCASE (*str*)
LEFT (*str1, count*)
LENGTH (*str*)
LOCATE (*str1, str2, [start]*)
LTRIM (*str*)
REPEAT (*str1, count*)
REPLACE (*str1, str2, str3*)
RIGHT (*str1, count*)
RTRIM (*str*)
SPACE (*count*)
SUBSTRING (*str, start, length*)
UCASE (*str*)

Data i czas

CURRENT_DATE ()
CURRENT_TIME ()
DAYNAME (*date*)
MONTH (*date*)
MONTHNAME (*date*)
QUARTER (*date*)
WEEK (*date*)
YEAR (*date*)
HOUR (*time*)
MINUTE (*time*)
SECOND (*time*)

Przykładowe funkcje numeryczne

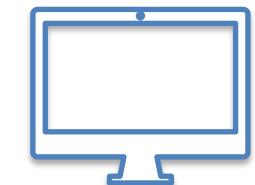
ABS / DEGREES / MOD / PI / POWER /
RAND / ROUND / TRUNCATE

Sekwencje escape

Sekwencja	Opis	Przykład
{e <number>}	Separator dziesiętny	MSAccess: {e 123.7} -> 123,7
{d <date>}	Data w formie tekstowej	Oracle: {d 2004-08-30} -> TO_DATE('2004-08-30', 'yyyy-mm-dd')
{t <time>}	Czas w formie tekstowej	SQL Server: {t 14:30:00} -> CONVERT(DATETIME, '14:30:00', 114)
{dt <date & time>}	j.w.	
{l <boolean>}	Stała logiczna	{l False} -> 0
{s <string>}		{s Company '1st Coding'} -> 'Company 1st Coding'
{id <identif>}	Identyfikator	Oracle: {id Order Details} -> "Order Details" MSSQL: {id Order Details} -> [Order Details]



FireDAC



Odzyskiwanie połączenia, tryb offline

Utrata połączenia z serwerem

Jak to działa?

- Operacje: Open, ExecSQL, Ping, Fetch, ...
- W momencie utraty połączenia
 - Wyjątek *EFDDBEngineException (Kind = ekServerGone)*

Wsparcie

- Pełne serwery RDBMS
 - Nie działa na *SQLite, Ms Access, Advantage DB*
 - Nie działa przez *ODBC i dbExpress*

Odzyskiwanie połączenia

- Automatyczne
- Ręczne = praca w trybie Offline

Włączenie (opcje)

- `ResourceOptions.AutoReconnect` (*default: False*)
- Zdarzenia w `TFDConnection`
 - *OnRecover: Co zrobić w momencie zerwania?*
 - Powtóż, Zrezygnuj, Przejdź w tryb Offline
 - *OnLost: Nie udana próba odzyskania*
 - *OnRestored: Udana próba odzyskania połączenia*

Utrzymywanie zbiorów danych

- W momencie odzyskiwania połączenia zbiory danych i samo połączenie nie jest zamknięte

FDConnection.OnRecover

```
procedure TForm1.FDConnection1Recover(ASender: TObject;
  const AInitiator: IFDStanObject; AException: Exception;
  var AAction: TFDPhysConnectionRecoverAction);
var
  iRes: Integer;
begin
  iRes := MessageDlg('Connection is lost. Offline - yes, Retry - ok, Fail -
Cancel',
    mtConfirmation, [mbYes, mbOK, mbCancel], 0);
  case iRes of
    mrYes:   AAction := faOfflineAbort;
    mrOk:    AAction := faRetry;
    mrCancel: AAction := faFail;
  end;
  Log('Connection is recovering');
end;
```

Tryb offline

Bezpołączeniowy

- Połączenie: zamknięte
- Zbiory danych: otwarte

Wykorzystanie

- Optymalizacja zasobów
- Słabe łącza
- Odzyskiwanie połączenia

Otwarte zbiory danych

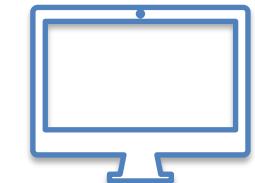
- Opcja: FetchOptions AutoFetchAll
 - Fetch All, Truncate, Disable (wyjątek)

Kontrola trybu Offline

- FDConnection.Offline
- FDConnection.Online
- Uwaga:

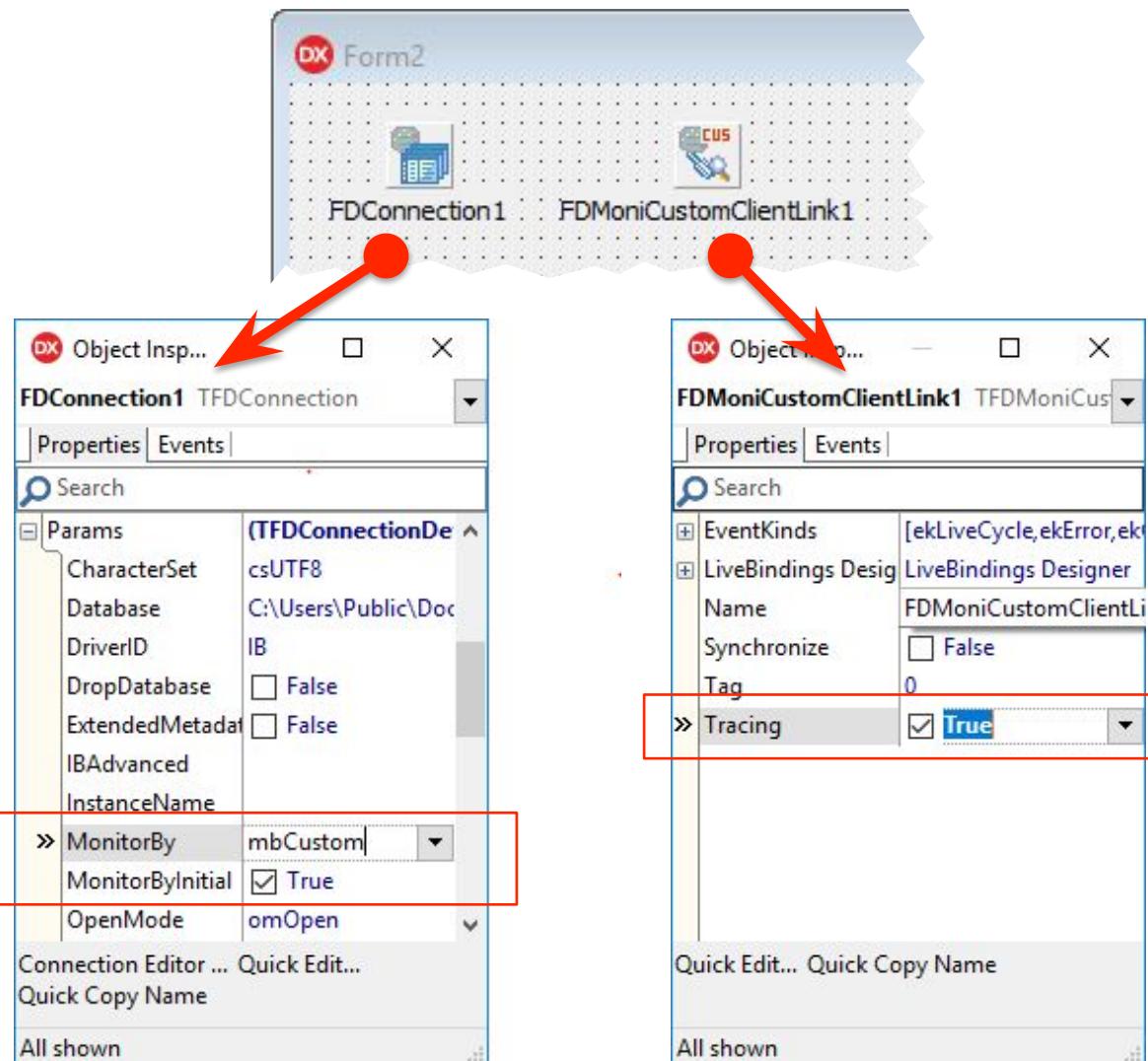


FireDAC



Diagnostyka połączenia

Diagnostyka FireDAC



Monitorowanie przez zdarzenie

```
procedure TForm1.FDMoniCustomClientLink1Output (
  ASender: TFDMoniClientLinkBase;
  const AClassName, AObjName, AMessage: string);
var
  m: string;
begin
  m := ConvertNewLineIntoSpaces(AMessage);
  if m.IndexOf('>> Open') >= 0 then
    ListBox1.Items.Add(m);
end;
```

Parametry zdarzenia:

- **AClassName** - nazwa klasy np.: *TFDConnection*
- **AObjName** - nazwa obiektu np.: *FDQuery1*

Monitorowanie - komponenty

Działanie

- Komponenty FireDAC przekazują
 - *Informacje diagnostyczne*

Komponenty

- **TFDMoniFlatFileClientLink**
 - *Monitorowanie do pliku*
- **TFDMoniRemoteClientLink**
 - *Monitorowanie przez ICP/IP*
 - *Komunikaty wysyłane do programu FDMonitor*
 - *FDMonitor - serwer nastuchujący na wybranym porcie*
- **TFDMoniCustomClientLink**
 - *Komunikaty przekazywane do zdarzenia*



FDMoniFlatFileClientLink1



FDMoniRemoteClientLink1



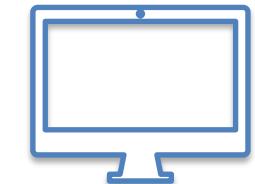
FDMoniCustomClientLink2

Monitorowanie z kodu

```
FDMoniCustomClientLink1.Tracing := True;  
FDConnection1.Params.Values['MonitorBy'] := 'Custom';  
FDConnection1.Connected := True;  
FDConnection1.ExecSQLScalar (  
    'select count(CompanyName) from {id Customers}') ;  
// Disable monitoring  
FDConnection1.ConnectionIntf.Tracing := False;  
// Enable monitoring  
FDConnection1.ConnectionIntf.Tracing := True;
```



FireDAC



Pobieranie metadanych

Pobieranie metadanych

Pobieranie nazw obiektów

- Metody Get**Names
 - *Wynikiem jest lista tekstów*
 - *Zawiera tylko nazwy obiektów*

Komponent

- TFDMetaInfoQuery
 - *Bardziej szczegółowe informacje*

Metoda	Opis
GetCatalogNames	Lista katalogów
GetSchemaNames	Lista schematów dla wybranego katalogu
GetTableNames	Lista tabel dla wybranego katalogu i schematu
GetFieldNames	Lista pól dla wybranej tabeli
GetKeyFieldNames	Lista pól klucza głównego dla wybranej tabeli
GetGeneratorNames	Lista generatorów dla katalogu i schematu
GetIndexNames	Lista indeksów dla katalogu i schematu
GetPackageName	Lista pakietów dla katalogu i schematu
GetStoredProcedureNames	Lista procedur dla katalogu, schematu i pakietu

Pobieranie metadanych GetTableNames

```
FDConnection1.GetTableNames(
  'Northwind', 'dbo', '',
  Memo1.Lines );
```

```
sl := TStringList.Create;
FDConnection1.GetTableNames(
  '', // katalog bieżący
  '', // schemat bieżący
  '', // maska nazw, pusta
  sl,
  [osMy, osSystem, osOther],
  [tkTable, tkView]);

// *** Wybranie obiektów, które nie
// *** zawierają tabel testowych
sTablesAndViews := '';
for sObjectName in sl do
  if
    not(sObjectName.Contains('FDQA_'))
  ) then
    sTablesAndViews :=
      StrAppendWithSeparatorAndSpace(
        sTablesAndViews, sObjectName,
        '');
sl.Free;
```

Metadane w FireDAC

Pobierane jednorazowo

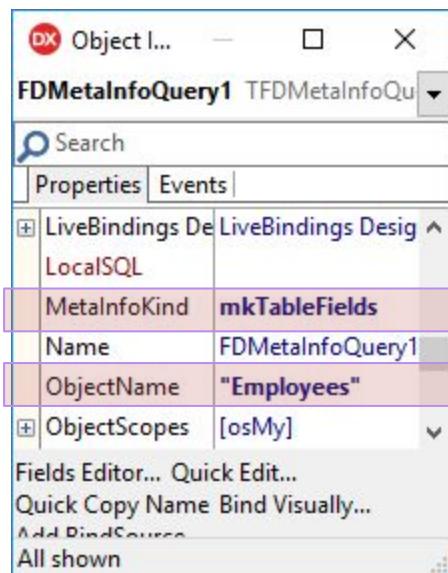
- Metadane są cache'owane
 - Pobrane raz i używane wielokrotnie
 - W ramach połączenia
 - Dwa połączenia mają zapamiętane różne metadane

Możliwe jest odświeżanie

- Odświeżanie metadanych
 - Wymaga wyłączenia cache'owania (opcje Fetch FireDAC'a)
 - TFDConnection
 - RefreshMetadataCache

TFDMetaInfoQuery

Komponent zwraca metadane w formie tabelarycznej jako TDataSet



The screenshot shows a Delphi IDE form titled 'Form1' containing the following components:

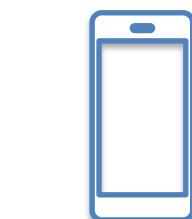
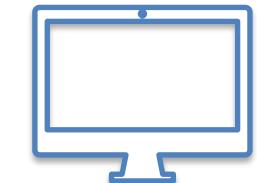
- FDConnection1
- FDMetaInfoQuery1 (selected)
- DataSource1

A red arrow points from the 'Properties' tab of the Object Inspector to the FDMetaInfoQuery1 component on the form. To the right of the form is a preview table showing the metadata for the 'Employees' table:

No.	Table	Column	Type	Precision	Scale	Length
1	Employees	EMPLOYEEID	INTEGER	0	0	
2	Employees	LASTNAME	VARCHAR	0	20	
3	Employees	FIRSTNAME	VARCHAR	0	10	
4	Employees	TITLE	VARCHAR	0	30	
5	Employees	TITLEOFCOURTESY	VARCHAR	0	25	
6	Employees	BIRTHDATE	DATE	0	0	
7	Employees	HIREDATE	DATE	0	0	
8	Employees	ADDRESS	VARCHAR	0	60	
9	Employees	CITY	VARCHAR	0	15	
10	Employees	REGION	VARCHAR	0	15	
11	Employees	POSTALCODE	VARCHAR	0	10	
12	Employees	COUNTRY	VARCHAR	0	15	
13	Employees	HOMEPHONE	VARCHAR	0	24	
14	Employees	EXTENSION	VARCHAR	0	4	
15	Employees	PHOTO	BLOB SUB_TYPE 0	0	0	
16	Employees	NOTES	BLOB SUB_TYPE 1	0	0	
17	Employees	REPORTSTO	INTEGER	0	0	
18	Employees	PHOTOPATH	VARCHAR	0	255	



FireDAC



FDS*Script*

Komponent TFDScript

Dlaczego?

- Może zawierać wiele poleceń SQL
- Można wykonać wszystkie polecenia SQL
- Może składać się z wielu transakcji
- Może zawierać dodatkowe polecenia poza językiem SQL
- Można podzielić go na kilka pod-skryptów
- Wykonanie jest w pełni kontrolowane przez aplikację
- Można logować kroki wykonania i wyniki
- Można oszacować postęp wykonania



Rozpoznawane dialekty

TFDScript ma specjalną obsługę

- Oracle **SQL*Plus**
- Microsoft **ISQL/OSQL**
- MySQL **mysql.exe/mysqldump.exe**
- Firebird/InterBase **ISQL**

Przykład skryptu Firebird'a

```
SET SQL DIALECT 3;
SET NAMES UTF8;
SET CLIENTLIB 'C:\fb25\bin\fbclient.dll';
CREATE DATABASE 'E:\Test2.ib'
    USER 'sysdba' PASSWORD 'masterkey'
    PAGE_SIZE 16384
    DEFAULT CHARACTER SET NONE;

SET TERM ^ ;
CREATE PROCEDURE MY_PROC RETURNS (aParam INTEGER) AS
BEGIN
    aParam = 10;
END^
```

Wykonanie skryptu

Stworzenie skryptu w kodzie i uruchomienie

```
with FDScript1 do begin
    SQLScripts.Clear;
    SQLScripts.Add;
    with SQLScripts[0].SQL do begin
        Add('INSERT INTO Brands VALUES (1, ''Audi'')');
        Add('INSERT INTO Brands VALUES (2, ''BMW'')');
    end;
    ValidateAll;
    ExecuteAll;
end;
```

Uruchomienie skryptu z pliku

```
with FDScript1 do begin
    SQLScriptFileName := 'c:\create.sql';
    ValidateAll;
    ExecuteAll;
end;
```

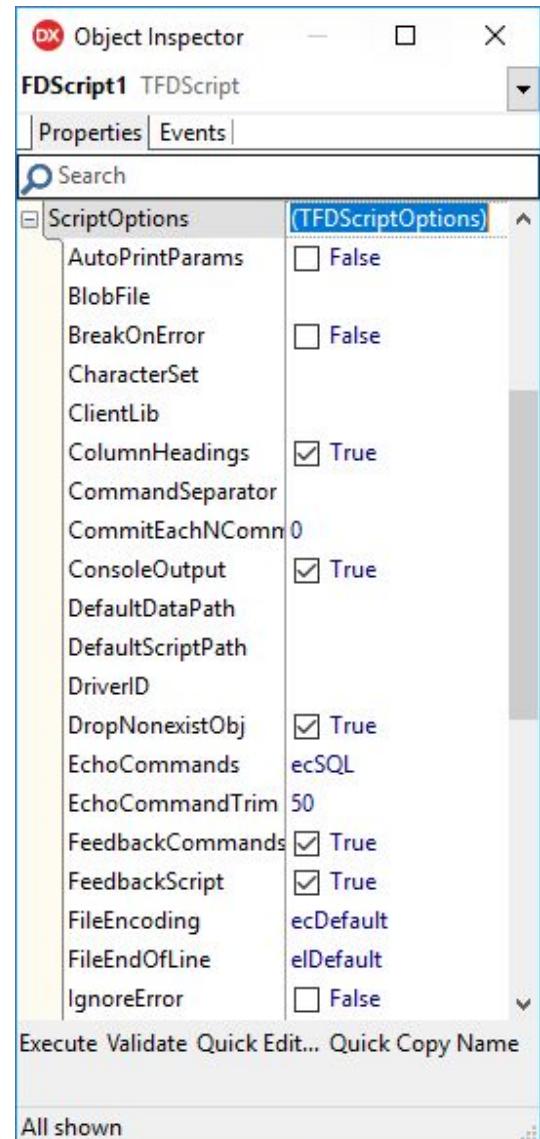
Opcje skryptu

TFDScript.ScriptOptions

- BLOB File
- BreakOnError
- CommandSeparator
- EchoCommands
 - *[ecNone, ecSQL, ecAll]*
 - *Nic, tylko SQL i PL/SQL, wszystko*
- FeedbackCommands
- MacroExpand: **True/False**
 - *TFDScript.Macros*

Ustawianie opcji

- Przez własność
- Polecienniem SET w skrypcie (*)

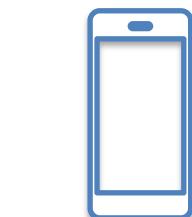
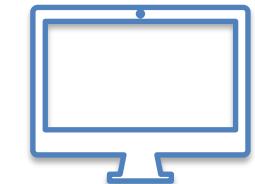


Podział skryptu na mniejsze pod skrypty

```
FDScript1.SQLScripts.Clear;
Script1 := FDScript1.SQLScripts.Add;
Script2 := FDScript1.SQLScripts.Add;
Script3 := FDScript1.SQLScripts.Add;
with Script1 do begin
  Name := 'root';
  SQL.Add('@first'); // explicitly call 'first' script
  SQL.Add('@second'); // explicitly call 'second' script
end;
with Script2 do begin
  Name := 'first';
  SQL.Add('create table t1 (...);');
  SQL.Add('create table t2 (...);');
end;
with Script3 do begin
  Name := 'second';
  SQL.Add('create procedure p1 (...);');
  SQL.Add('create procedure p2 (...);');
end;
FDScript1.ValidateAll;
FDScript1.ExecuteAll;
```



FireDAC



Array DML

- Tablicowe polecenia modyfikacji danych

Array DML

Co to jest?

- Array DML
- Modyfikacja (poprawa / dodawanie) dużych zbiorów danych z dużą wydajnością
- Polecenia SQL: INSERT i UPDATE

Wsparcie FireDAC

- Wykorzystuje ten tryb jeśli platforma go wspiera
- Emuluje działanie gdy nie wspiera

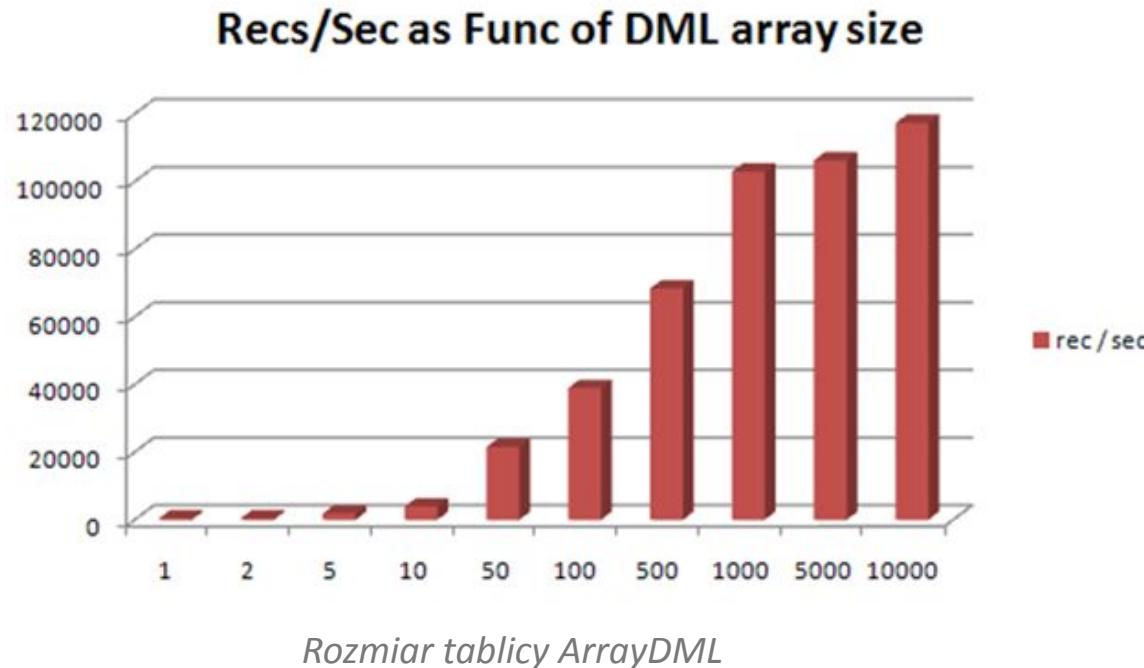
ArrayDML

```
FDQuery1.SQL.Text := 'INSERT INTO ADQA_Batch_test '+  
  '(tint, tstring) VALUES (:f1, :f2)';  
FDQuery1.Params.ArraySize := 100;  
for i := 0 to 100-1 do begin  
  FDQuery1.Params[0].AsIntegers[i] := i;  
  FDQuery1.Params[1].AsStrings[i] := GenerateRandomWord();  
end;  
FDQuery1.Execute(100, 0);  
ShowMessage(IntToStr(FDQuery1.RowsAffected));
```

Wydajność Array DML

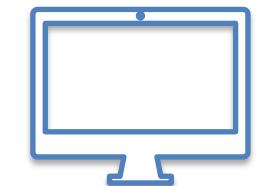
100 000 prostych poleceń INSERT

```
INSERT INTO ADQA_Batch_test (tint, tstring) VALUES (:f1, :f2)
```





FireDAC



Transakcje

- Praca z transakcjami
- Zagnieżdżanie transakcji

Transakcja

Opis

- Zbiór operacji wykonanych poprawnie w całości lub w ogóle

Polecenia

- Rozpocznij transakcję
- Zatwierdź transakcję
- Odrzuć transakcję

SQL

- START / BEGIN TRANS
- COMMIT
- ROLLBACK

Rozpocznij transakcję

Operacja 1

Operacja 2

Operacja 3

Operacja 4

Zatwierdź transakcję

Transakcje w FireDAC

```
FDConnection1.StartTransaction;  
try  
    FDQuery1.ExecSQL;  
    ....  
    FDQuery1.ExecSQL;  
    FDConnection1.Commit;  
except  
    FDConnection1.Rollback;  
    raise;  
end;
```

```
FDConnection1.StartTransaction;  
try  
    FDQuery1.ExecSQL;  
    ....  
    FDConnection1.StartTransaction;  
    try  
        FDQuery1.ExecSQL;  
        ....  
        FDConnection1.Commit;  
    except  
        FDConnection1.Rollback;  
        raise;  
    end;  
    FDConnection1.Commit;  
except  
    FDConnection1.Rollback;  
    raise;  
end;
```

ACID

ATOMIC - atomowa

- Operacja jest pojedynczą jednostką
- Wykonywane jest wszystko albo nic.

CONSISTENT - spójna

- Zakończona transakcja pozostawi bazę danych w stanie spójnym

ISOLATED - izolowana

- Dwie lub więcej równoczesnych transakcji zostaną wykonane w izolacji
- Jedna transakcja nie wpłynie negatywnie na drugą

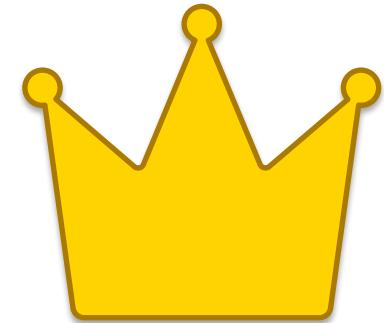
DURABILITY - trwała

- Częściowe wyniki transakcji są przechowywane trwale

Poziomy izolacji transakcji

Poziomy

1. Read uncommitted
 - *niezatwierdzone zmiany*
2. Read committed
 - *Zatwierdzone zmiany*
3. Repeatable read
 - *W kolejnych odczytach ta sama wartość*
4. Serializable / Snapshot
 - *Pełna izolacja*



Błędy izolacji

Brudne odczyty

- Zmiany wykonane w ramach innych niezatwierdzonych transakcji są widoczne

Powtarzające się odczyty

- Dane odczytywane za drugim i kolejnym razem mogą zostać zmienione przez inną zatwierdzoną transakcję

Fantomy

- Dane mogą ulec zmianie w czasie trwania transakcji jeśli ich nie odczytaliśmy

Poziom	Brudne odczyty	Nie powtarzające się odczyty	Fantomy
Read uncommitted			
Read committed	NIE		
Repeatable read	NIE	NIE	
Serializable	NIE	NIE	NIE

Optymalizacja transakcji

Długość

- Jak najkrótsze transakcje



Rollback

- Unikanie, jest bardzo kosztowny
- FireDAC: (Centralized) Cached Updates
- Używanie pamięciowych zbiorów danych

Poziom

- Izolacja jest bardzo kosztowna
- Im silniejsza tym kosztowniejsza
- Im niższy poziom tym lepiej
- Zazwyczaj optymalny poziom: **read committed**

Transakcje Ciągłe

Nie zamyka zbiorów danych

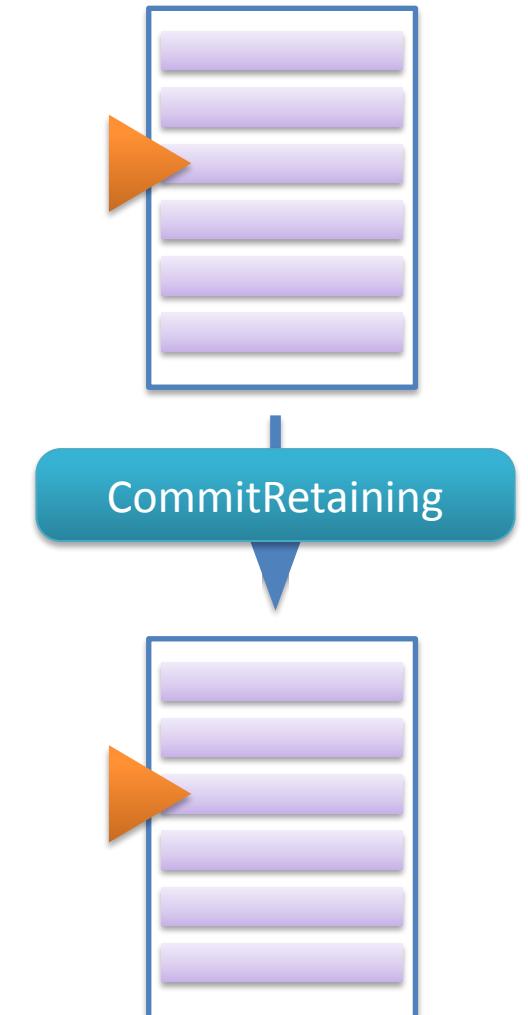
- Zatwierdzenie / wycofanie transakcji nie zamyka zborów danych z nią związanych

Metody TFDConnection

- CommitRetaining / RollbackRetaining

Realizacja

- Interbase/Firebird
 - *Realizowane przez serwer*
 - *Specyficzna funkcjonalność IB/FB*
- Inne serwery
 - *Emulowane przez FireDAC*



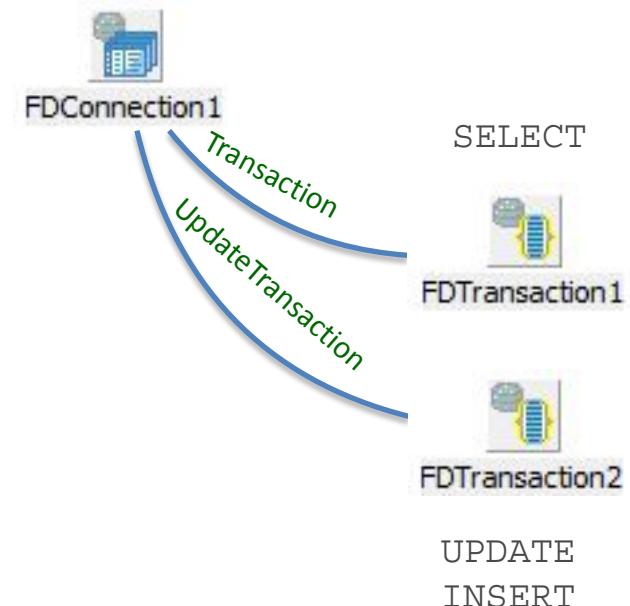
Transakcje Równoległe

Specyficzne

- Funkcjonalność IB / FB
- Dostępna dla innych platform

Komponenty

- TFDTransaction
- TFDConnection / TFDQuery
 - *Transaction*
 - *UpdateTransaction*



Transakcje równoległe

TFDQuery TFDTransaction



```
begin
  // transaction for update: read_committed, rec_version, nowait
  UpdateTransaction.Connection := FDConnection1;
  FDConnection1.UpdateOptions.LockWait := False;
  UpdateTransaction.Options.ReadOnly := False;
  UpdateTransaction.Options.Isolation := xiReadCommitted;

  ReadTransaction.Connection := FDConnection1;
  ReadTransaction.Options.ReadOnly := True;
  ReadTransaction.Options.Isolation := xiReadCommitted;

  SelectQuery.Transaction := ReadTransaction;
  SelectQuery.UpdateTransaction := UpdateTransaction;
end;
```

Sekcja



FireDAC

FireDAC

- *Uniwersalny*
- *Wydajny*

Uniwersalny = niezależny od platformy

Wiele platform

- Aplikację można podłączyć do kilku różnych (przynajmniej dwóch) platform DBMS, np.: Oracle, MS SQL Server i SQLite

Zapytania

- Dostosowane do platformy - baza wiedzy
- Preprocesor SQL
- Generowanie UPDATE'ów i INSERT'ów

Połączenie

- Obsługa transakcji i błędów
- Weryfikacja metadanych

Dane

- Uniwersalne typy
- Pola auto-numerowane i sekwencje
- Układ master-detail

Unifikacja

Typy danych

Transakcje

Preprocesor SQL

Master-detail

Generowanie UPDATE'ów

Zdarzenia serwera

Pola auto-numerowane

Transfer danych

Raportowanie błędów

Metadane

Stronicowanie i zliczanie
zbiorów danych

Wydajność

Pobieranie danych

- Duże tabele: oszczędna i szybka praca
- Ograniczanie zbioru danych
- Kursor jednokierunkowy

Polecenia

- Szybkie UPDATE'y
- Grupowanie poleceń
- Tablicowe polecenia DML

Praca równoległa

- Wielowątkowość
- Asynchroniczne polecenia
- Pula połączeń

Buforowanie metadanych

Wydajność

Live Data Window

Grupowanie poleceń

Optymalizacja pobierania
wierszy

Wykonanie asynchroniczne

Buforowanie metadanych

Przerywanie długich poleceń

Pula połączeń

Szybkie UPDATE'y

Array DML

Sekcja



Zbiór danych w FireDAC

FireDAC

- TDataSet - powtórka
- Kursor FireDAC a kursor na serwerze
- FDTable
- Pobieranie wierszy
- Przygotowanie zapytania
- Parametry SQL

Sekcja



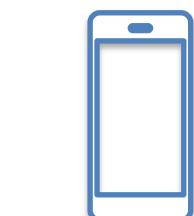
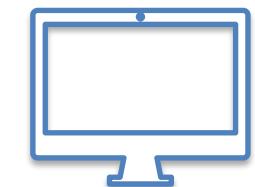
... ciąg dalszy ...

FireDAC

- Aktualizacja danych
- Widoki: sortowanie i filtrowanie wierszy
- Wyszukiwanie rekordów



FireDAC



TDataSet

powtórka wiadomości

- Przypomnienie wiadomości

Zbiór danych – charakterystyka

Struktura

- tabelaryczna
- kolumny mają ustaloną nazwę i typ

Pola

- bieżący wiersz (rekord)
- dostęp tylko do bieżącego wiersza

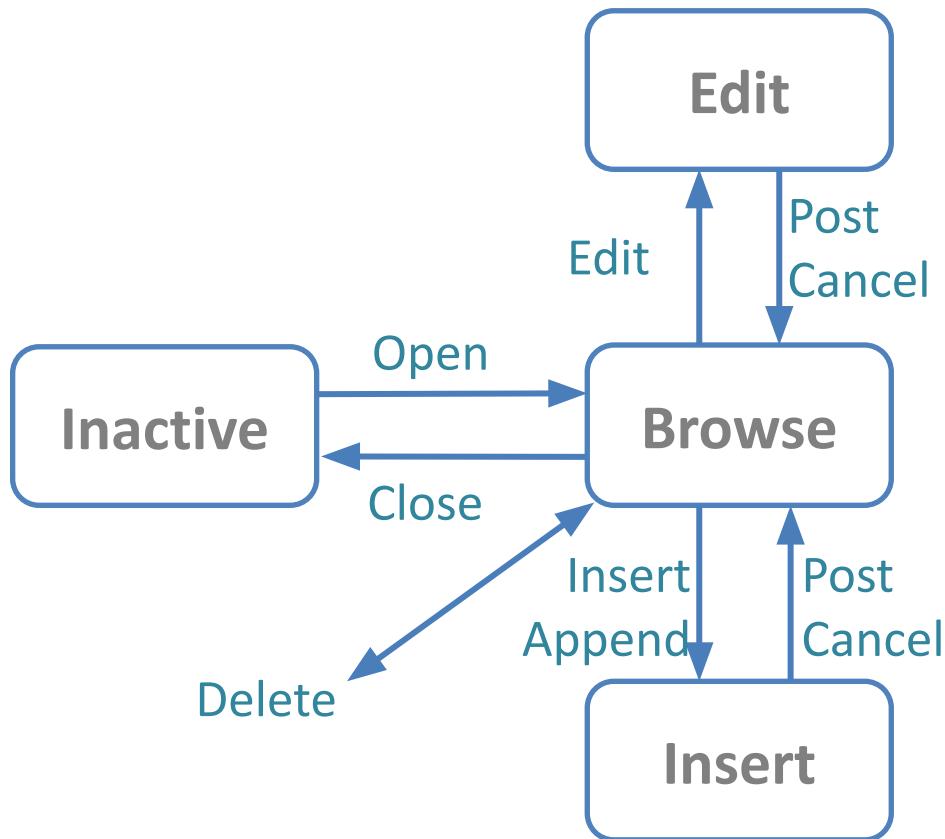
Nawigacja

- poprzedni – następny
- początek – koniec
- wyszukiwanie

ID	Imię	Nazwisko
Integer	String(20)	String(30)
1	Bogdan	Polak
2	Jan	Kowalski
3	Anna	Świeradowska
6	Daniel	Konarowski
9	Sylwia	Kamińska
13	Robert	Mariański

Prev  Next 

Stan zbioru danych



Tryb edycji

Umożliwia zmianę wartości pól.
Konieczne: zatwierdzenie zmian
metodą Post

Tryb przeglądania

Podstawowy tryb dla otwartego
zbioru danych. Nie pozwala na
dodawanie i modyfikację danych

Tryb dodawania

Przejście do niego inicjuje nowy
rekord (puste pola), który dopisany
zostanie po zatwierdzeniu Post-em

Dostęp do wartości pól z kodu

Dostęp do wartości pola

```
dsPeople.Fields[Index].Value := 'Poland';  
dsPeople.FieldValues['Country'] := 'Poland';  
dsPeople['Country'] := 'Poland';  
dsPeople.FieldName('Country').Value := 'Poland';  
dsPeople.FieldName('Country').AsString := 'Poland';
```

pole

Dostęp obiektowy

```
dsPeopleCountry.Value := 'Poland';
```

Podstawowe zdarzenia zbioru danych

Zmiana stanu

- Before Open After Open
- Before Close After Close
- Before Edit After Edit
- Before Insert After Insert
- Before Post After Post
- Before Cancel After Cancel

Podstawowe

- After/Before Delete *usuwanie rekordu*
- After/Before Refresh
- After/Before Scroll *zmiana pozycji kurSORA*
- *OnCalcFields* *wyliczenie wartości pól wyliczeniowych*
- *OnFilterRecord* *warunek filtrowania*
- OnNewRecord *wartości domyśLne*

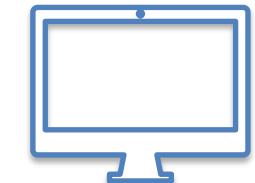
Przeliczenia w kodzie na zbiorze

```
Bookmark := myDataSet.GetBookmark;      // stworzenie zakładki
myDataSet DisableControls;

try
    myDataSet.First;
    while not myDataSet.Eof do begin
        // **** tutaj wstaw przetwarzanie rekordów
        // ... myDataSet.FieldName('fieldname').Value
        myDataSet.Next;
    end;
finally
    if myDataSet.BookmarkValid(bookmark) then
        myDataSet.GotoBookmark(Bookmark);
    myDataSet.FreeBookmark(Bookmark);
    myDataSet.EnableControls;
end;
```

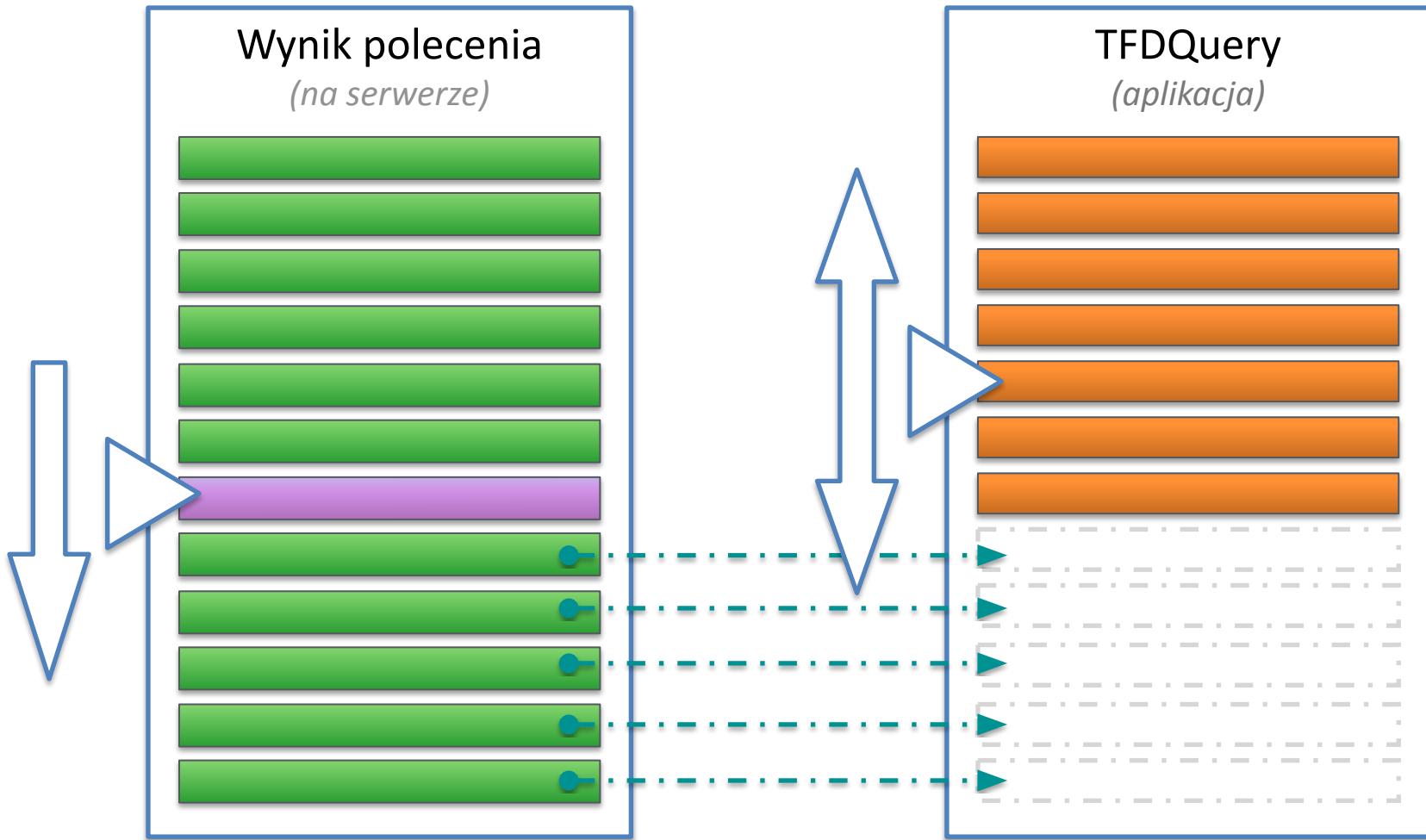


FireDAC

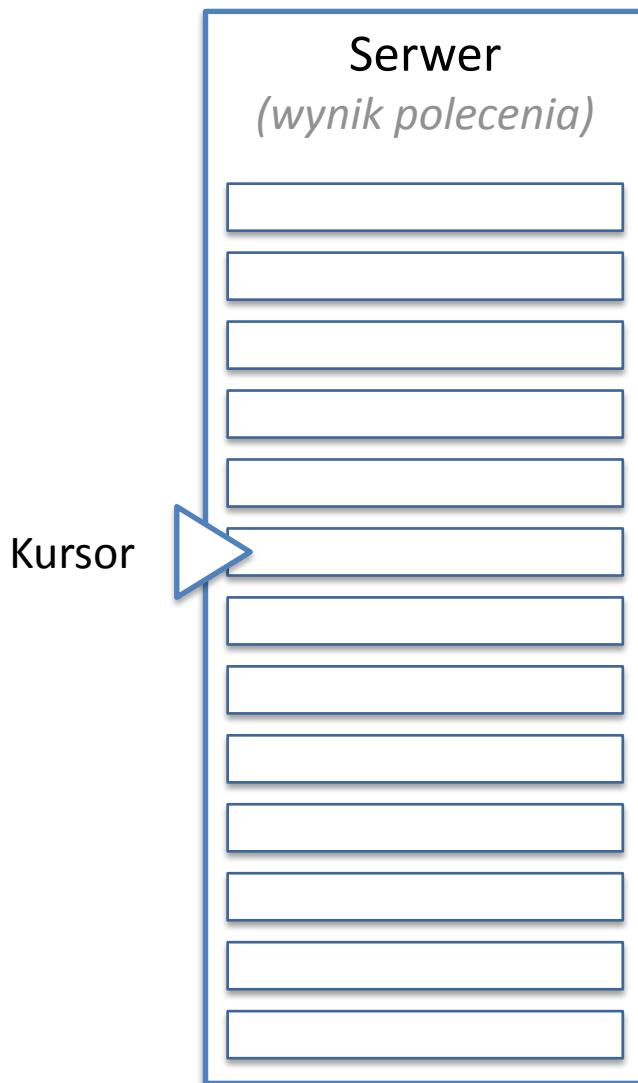


Kursor FireDAC a kurSOR na serwerze

Kursor na serwerze a zbiór danych



Tryb unidirectional



Tylko bieżący wiersz (aktualnie wskazywany przez kurSOR na serwerze) jest przechowywany w pamięci aplikacji.

Tylko przejście na kolejny wiersz (Next)

TFDTable
TFDQuery
Unidirectional

```
FDQuery1.FetchOptions.Unidirectional := True;
```

Tryb unidirectional

Ustawienie

- `FetchOptions.Unidirectional := True`

Zalety

- Szybkość
- Najmniejsze zużycie zasobów

Wady

- Nie można wykorzystać do pracy z `TDBGrie'iem`
- Dlaczego?



Tryby pracy

Standardowy

- Podobnie jak TQuery
- Generuje 1 polecenie SELECT i pobiera dane zgodnie z przyjętą strategią pobierania

Unidirectional

- Minimalne zasoby
- Zbiór ForwardOnly
- Nie można nawigować i połączyć do TDBGrid'a

Live Data Window

- Dwukierunkowa praca na bardzo dużych zbiorach danych
- Optymalizacja, która pozwala pobierać minimalną ilość danych

Kursor na serwerze

Obsługa kurSORA

- Kiedy serwer rozpoczyna przetwarzanie polecenia zwracającego wiersze to zakłada kursor

Zmiana rodzaju kurSORA

- Opcja
- FetchOptions.CursorKind
 - *ckAutomatic, ckDefault, ckDynamic, ckStatic, ckForwardOnly*

Rodzaje kurSORów na serwerze

ckAutomatic

- FireDAC sam dobiera optymalny rodzaj

ckDefault

- Domyślny rodzaj proponowany przez bibliotekę kliencką

ckDynamic

- Porusza się po kluczu głównym i dynamicznie doładowuje dane
- Dane w tabeli mogą się zmienić w czasie nawigacji

ckStatic

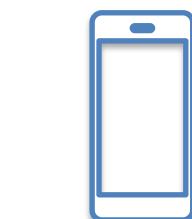
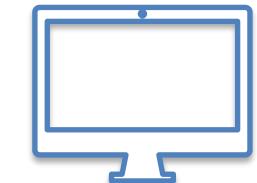
- Statyczne dane. Nie mogą się zmienić w czasie pobierania

ckForwardOnly

- Kursor jednokierunkowy
- FireDAC pobiera dane tylko do przodu



FireDAC



TFDTable

Komponent TFDTable

Zoptymalizowany

- Pojedynczy **widok** lub tabela
 - Wraz z dodatkowymi polami *lookup, kalkulowanymi, itp.*
- Optymalna
 - Odwołanie do pojedynczego obiektu
 - Tryb Live Data Window (LDW)
- Łatwa w użyciu
 - Sortowanie, filtrowanie
- Indeksy
 - Pozwala wykorzystać indeksy utrzymywane przez serwer SQL



Otwarcie tabeli

Ustawienie

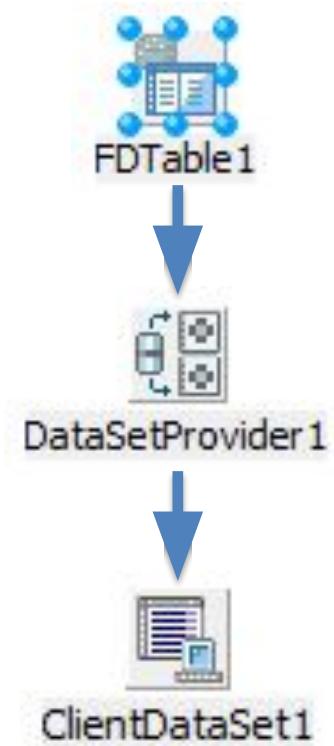
- TableName
- IndexName
 - *FetchOptions.Items* musi zawierać *fiMeta*
- IndexFieldNames
- Jednoznaczne identyfikatory
 - *FormatOptions.Quote := True*

```
FDTable1.TableName := 'CUSTOMERS';  
FDTable1.IndexFieldNames := 'CustNo';  
FDTable1.Open;
```

TFDTable z TClientDataSet

Uwaga! Dotyczy migracji starych projektów do FireDAC

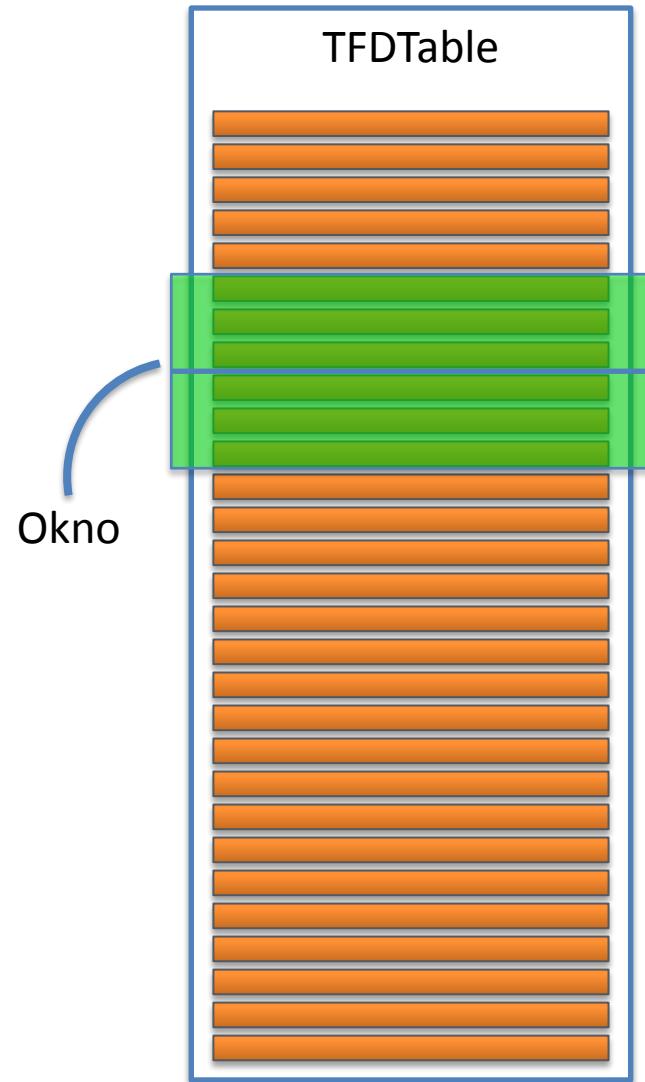
- Układ
 - *TFDTable + TDataSetProvider + TClientDataSet*
- TFDTable
 - *Może od nowa otwierać cały zbiór przy każdej modyfikacji*
- Trzeba ustawić
 - *FetchOptions.Unidirectional := True*
 - *FetchOptions.AutoClose := False*
 - *Przed otwarciem komponentu TClientDataSet*



Tryb LDW = Live Data Window

Ograniczony zbiór danych

- 2x FetchOptions.RowsetSize
- Tylko potrzebne dane
 - *W momencie wyjścia poza zakres okna generowane są nowe polecenia SELECT pobierające tylko potrzebne dane*
- Skok na koniec
 - *Nie powoduje załadowania wszystkich danych*
- Locate
 - *Nie powoduje załadowania wszystkich danych pomiędzy*



Analiza trybu LDW

Zasoby

- Minimalizuje zużycie pamięci aplikacji
- Tylko trochę większe zużycie niż w trybie unidirectional

Szybkość

- Nie ma opóźnień na pobieranie wielu wierszy

Obciążenie serwera

- Mniejszy ruch
- Użycie indeksów

Łatwy w implementacji

- Nie wymaga żadnych specjalnych umiejętności od programisty. Komponent automatycznie rozpoznaje.

Aktualne dane

Wymagania dla trybu LDW

Klucz unikalny lub klucz główny

- Tabela musi mieć zdefiniowany klucz główny lub dowolną kolumnę jako unikalną

Zgodność porządku sortowania

- Porządek sortowania danych na serwerze (ang. sort collation) i porządek sortowania w aplikacji muszą być identyczne
- Jeśli tak nie będzie
 - Tryb LDW zostanie uruchomiony
 - W czasie nawigacji możemy dostawać wyjątki "unique key violation"

Parametry TFDTTable

- CachedUpdates: False
- FetchOptions.Unidirectional: False
- FetchOptions.CursorKind: ckAutomatic / ckDynamic
- (wartości domyślne są OK)

Filtrowanie i sortowanie w trybie LWD

Filtrowanie

```
dt := Trunc(MonthCalendar1.Date);
FDTable1.Filter := 'DateField = {d ' +
  FormatDateTime('yyyy-mm-dd', dt) + '}';
FDTable1.Filtered := True;
```

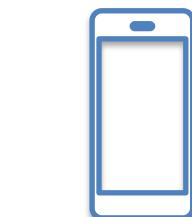
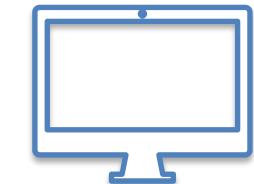
Wartość filtru zostanie
zamieniona na warunek w
klauzuli WHERE

Sortowanie (język polski)

```
FDTable1.FormatOptions.SortLocale := Windows.MAKELCID(
  MAKELANGID (LANG_POLISH, SUBLANG_POLISH_POLAND), SORT_DEFAULT);
// Use the punctuation and other symbols case insensitive sorting
FDTable1.FormatOptions.SortOptions := [soNoSymbols];
FDTable1.IndexFieldNames := 'NAME';
FDTable1.TableName := 'CUSTOMERS';
FDTable1.Open;
```



FireDAC



Pobieranie wierszy

Pobieranie wierszy: RowsetSize

Pobieranie paczkami

- Bardziej optymalne
 - *Jeśli zamierzamy je wyświetlać w formie tabelarycznej TDBGrid*
- Minimalizacja odwołań sieciowych do serwera
- Rozmiar paczki (w wierszach)
 - *Opcja FetchOptions.RowsetSize. - domyślnie 50*
- Duże wartości powodują spadek wydajności
 - *Powyżej 2000*

Uwaga na emulację

- Jeśli platforma bazodanowa nie wspiera tego trybu to
 - *RowsetSize := 1*

Pobieranie wierszy: Mode

FetchOptions.Mode

- fmOnDemand
 - *Kolejna paczka na żądanie*
- fmAll
 - *Pobierane są wszystkie wiersze na początku*
- fmManual
 - *Stronicowanie*
 - *programista: FetchNext, FetchAll*
- fmExactRecsMax
 - *FeatchAll ale liczba wierszy musi być odpowiednia*
 - *Jeśli liczba wierszy jest większa od FetchOptions.RecsMax to zwracany jest wyjątek*

Pobranie wszystkich wierszy

Własność

- TFDDataset.SourceEOF = True

Jeśli

- FetchOptions.AutoClose = True (*domyślna wartość*)
- polecenie związane ze zbiorem danych jest zwalniane
- Nie jest zamknięty sam zbiór

Pobranie jednej strony danych

Wersja A

- `FetchOptions.Mode = fmManual`

Wersja B

- `FetchOptions.RecsSkip`
- `FetchOptions.RecsMax`
- Jeśli platforma to wspiera to FireDAC zamienia te wartości na odpowiednie klauzule doklejane do zapytania SELECT
- Jeśli te opcje zostaną zmienione to trzeba wywołać
 - `TFDQuery.Unprepare`

Przykład RecSkip, RecMax

```
FDQuery1.FetchOptions.RecSkip := 0;  
FDQuery1.FetchOptions.RecMax := 100;  
FDQuery1.Open;  
// process rows  
  
FDQuery1.Disconnect;  
FDQuery1.FetchOptions.RecSkip := 100;  
FDQuery1.Open;  
// process rows  
  
FDQuery1.Disconnect;  
FDQuery1.FetchOptions.RecSkip := 200;  
FDQuery1.Open;  
// process rows
```

FetchAgain

```
// set SQL command
FDCommand1.CommandText := 'select * from orders where customerid = :cid';
// link command to table adapter
FDTAdapter1.SelectCommand := FDCommand1;
// link table adapter to memtable
FDMemTable1.Adapter := FDTAdapter1;

// set SQL command parameter value, then execute command and fetch all records
FDCommand1.Params[0].AsInteger := 100;
FDMemTable1.Open;
FDMemTable1.FetchAll;

// reset SourceEOF flag, reexecute command by hands and fetch all records
FDMemTable1.FetchAgain;
FDCommand1.Close;
FDCommand1.Params[0].AsInteger := 200;
FDCommand1.Open;
FDMemTable1.FetchAll;

// here we will have in memtable orders for customers with ID=100 and ID=200
```

Pozwala „dodatać” rekordy z kolejnego polecenia. Ustawia SourceEOF : False

Opóźnione pobieranie

Opcja FetchOptions.Items (zbiór)

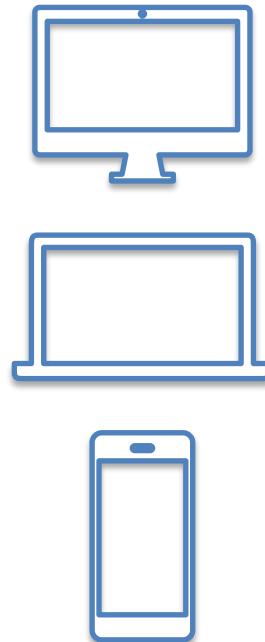
- Domyślnie: [fiDetails, fiBlobs, fiMeta]
- Wyłączenie fiBlobs: opóźnione ładowanie.
- Można je wymusić przez FetchBlobs (niezbędne w trybie fmManual)
- wyłączenie fiDetails: opóźnione ładowanie zagnieżdżonych zbiorów danych
- Wymuszenie pobierania: FetchDetails

Uwaga

- Przy opóźnionym ładowaniu BLOB'ów:
 - Jeśli biblioteka kliencka pobiera BLOB'y przez wartość to one zostaną pobrane przez klienta, ale nie załadowane do aplikacji (np.: Oracle pole LONG, MySQL, SQL Server, SQLite).
 - Jeśli biblioteka pobiera BLOB'y przez wskaźnik to nie zostaną dostarczone (np.: Oracle pola CLOB / BLOB, Interbase, Firebird)



FireDAC

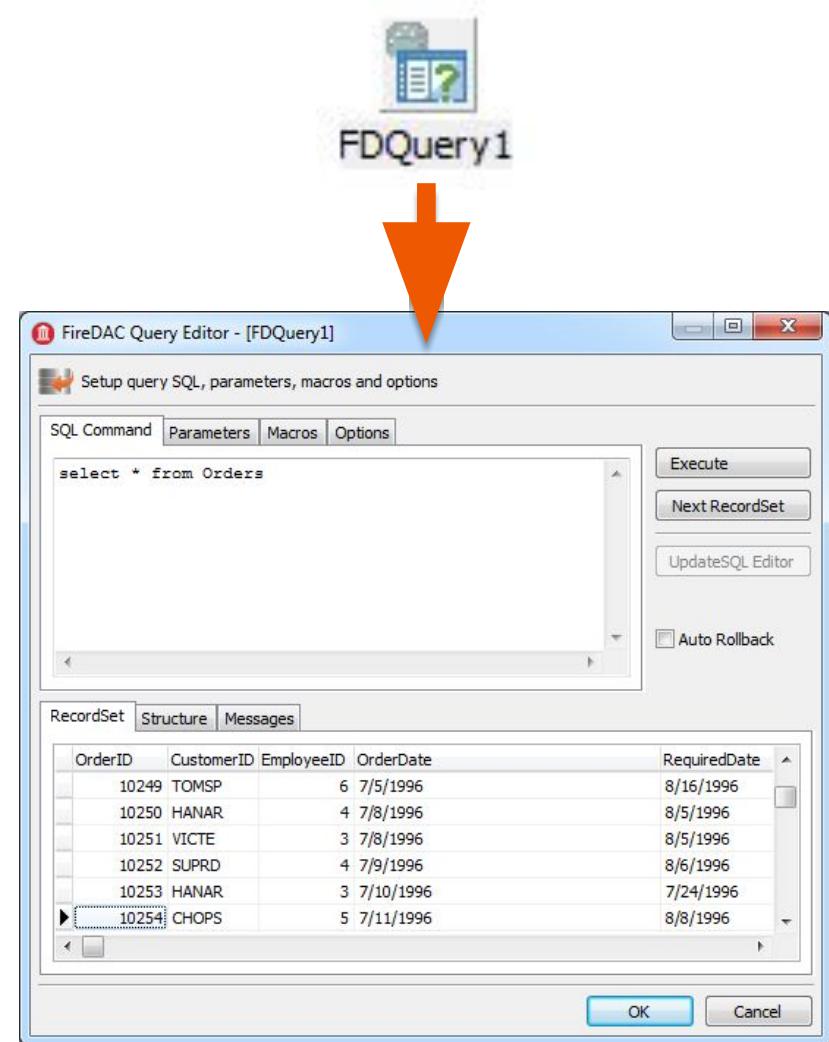


TFDQuery - przygotowanie i realizacja zapytania

TFQuery - Query Editor

W projektancie double-click na TFDQuery

- Wprowadzenie polecenia
- Można ustawić
 - parametry
 - wartości zmiennych dla makr
 - opcje dla komponentu
- Funkcjonalność okna
 - Wykonanie zapytania i weryfikacji wyników
 - Sprawdzenie struktury
 - Odczytanie wiadomości
 - Np. MS SQL Server



Jak działa TFDQuery?

TFDQuery

Polecenia SQL

SELECT

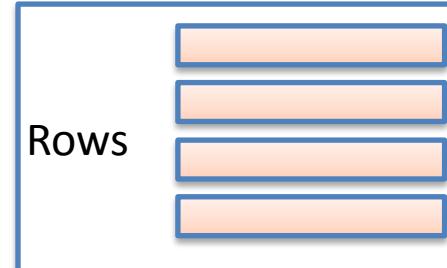
DELETE

INSERT

UPDATE

Table: TFDDatSTable

Rows



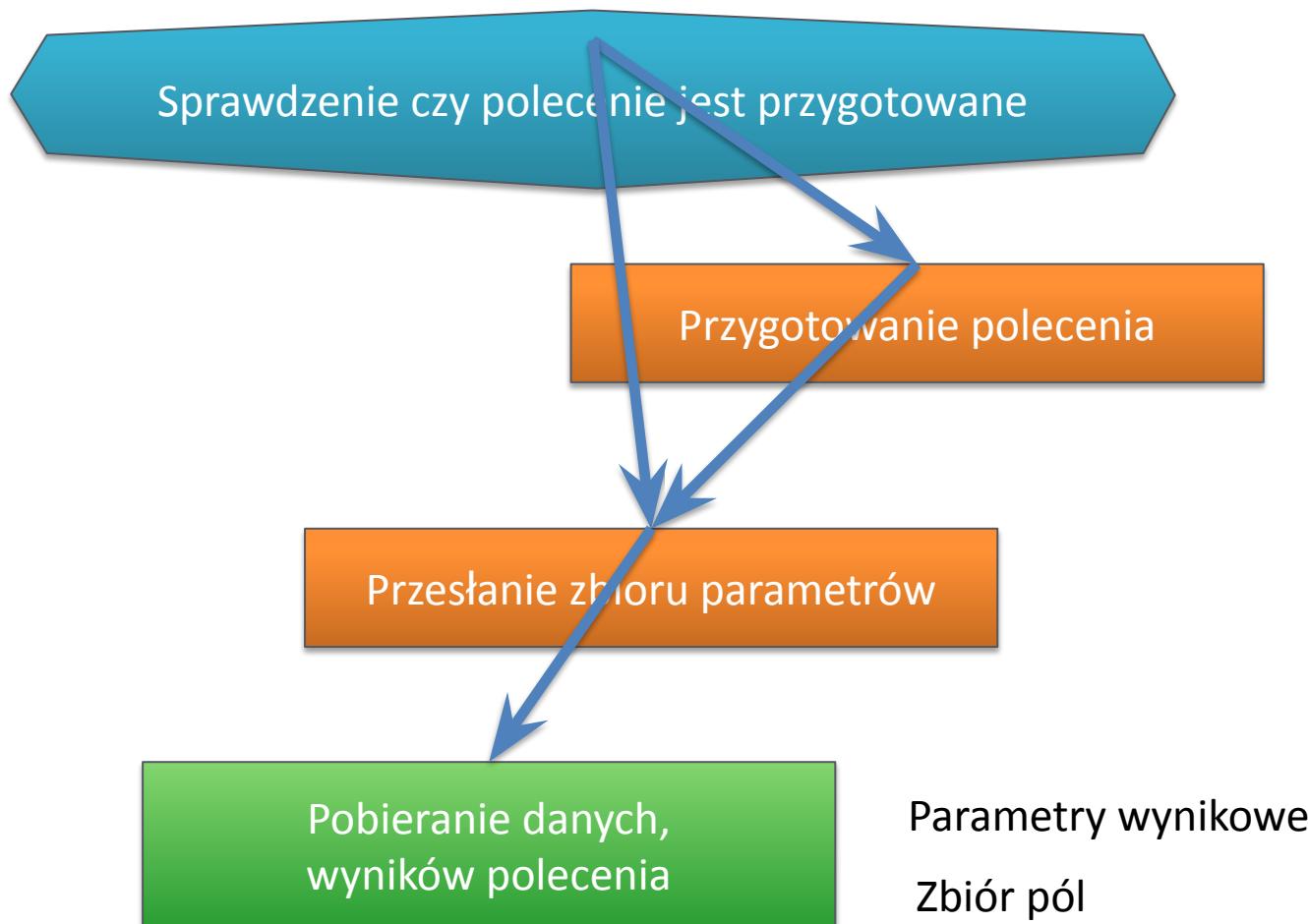
TFDDatSTable

Wewnętrzny zbiór danych TFDQuery: Table: TFDDatSTable

Użycie FDCommand i metody Fetch oraz praca na TFDDatSTable

```
// default value: CommandKind = skUnknown
FDCommand1.CommandKind := skSelect;
FDCommand1.CommandText.Text := 'select * from {id Customers}';
//      datS: FireDAC.DatS.TFDDatSTable;
datS := FDCommand1.Define();
FDCommand1.Open();
FDCommand1.Fetch(datS);
FDCommand1.Close();
for i := 0 to datS.Rows.Count-1 do
  ListBox1.Items.Add(
    datS.Rows[i].GetData('CompanyName') );
```

Wykonanie polecenia SQL



Przygotowanie zapytania

Kroki:

1. Połączenie

- *Ustawienie połączenia active / online*
- Weryfikacja parametrów
- Preprocesor FireDAC
 - *Przetwarzanie polecenia przez preprocesor FireDAC*
- (opcja) Zakres wierszy
 - *Modyfikacja zapytania na podstawie parametrów RecsSkip / RecsMax*
- (opcja IB/FB) Start transakcji
- Wysłanie polecenia
 - *Wysłanie wynikowego polecenia do serwera SQL*

Przygotowanie polecenia SQL - uwagi

Przygotowanie

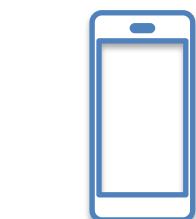
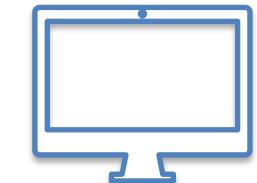
- Za pierwszym razem
 - Zawsze gdy otwieramy zapytanie, a wcześniej nie zostało przygotowane
- Wielokrotnie używane
 - Przy kolejnych wykonaniach wykorzystywane jest pierwsze przygotowanie
- Wywołanie
 - Prepare
 - Open
 - Active := True

Optymalizacja

- Unikać
 - Należy unikać przygotowania niepotrzebnych poleceń
 - **Prepare** rezerwuje zasoby na serwerze SQL
- Zwolnienie zasobów
 - **Unprepare** lub **Disconnect**



FireDAC



Parametry SQL i FireDAC

Parametry w Open

TFDQuery.Open(sql, params, types)

Parametry w poleceniach SQL

Automatyczne wypełnianie

- Wypełniana jest: tylko nazwa i pozycja parametru
- Uzupełnianie informacji o parametrach
 - Jeśli to jest istotne to typ i rodzaj parametru (In / Out / InOut) powinny być ustawione przez programistę

Dodawane manualnie

- Co trzeba ustawić?
 - Pozycja / nazwa / typ / rodzaj / wartość

Parametry - ważne uwagi

Czy używać parametrów?

- **Tak** i to jak najczęściej
- Dlaczego? Wydajność i administracja serwerem

Generowanie automatyczne / manualne

- TFDQuery.ResourceOptions.ParamCreate: True / False

Metoda łączenia parametrów

- Parametry są łączone na serwerze
- TFDQuery.Params.BindMode: pbByName / pbByNumber

Domyślne typ danych i rodzaj

- FormatOptions.DefaultParamDataType: ftUnknown
- ResourceOptions.DefaultParamType: ptInput

Użycie parametrów

Ustawienie wartości parametru

```
FDQuery1.SQL.Text := 'select * from  
tab where code = :Code';  
FDQuery1.ParamByName('code').AsString  
:= '123';  
FDQuery1.Open;
```

```
FDQuery1.SQL.Text := 'select * from  
tab where code = :Code';  
FDQuery1.Open('', ['123']);
```

Manualne tworzenie parametrów

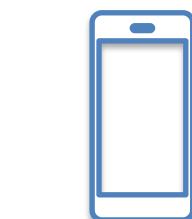
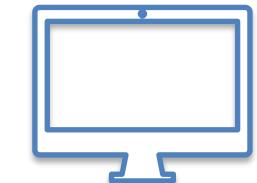
```
FDQuery1.ResourceOptions.ParamCreate  
:= False;  
FDQuery1.SQL.Text := 'select * from  
tab where code = :Code';  
with FDQuery1.Params do begin  
  Clear;  
  with Add do begin  
    Name := 'CODE';  
    DataType := ftString;  
    Size := 10;  
    ParamType := ptInput;  
  end;  
end;
```

Wartość parametru = NULL

```
with FDQuery1.ParamByName('name') do  
begin  
  DataType := ftString;  
  Clear;  
end;  
FDQuery1.ExecSQL;
```



FireDAC



Procedury składowane

Wykonanie procedur składowanych

Procedura

- Nazwa
- Parametry wejściowe
- Parametry wynikowe

Warianty zwracania wyników

- Jako zbiór danych
- Jako parametr OUT
- Wywołanie procedury przez TFDQuery (zależne od platformy)

Procedury składowane - wypełnianie parametrów

Automatyczne wypełnianie

```
FDStoredProc1.StoredProcName :=  
  'my_proc';  
FDStoredProc1.Prepare;
```

Ręczne wypełnianie

```
FDStoredProc1.StoredProcName :=  
  'my_proc';  
with FDStoredProc1.FetchOptions do  
  Items := Items - [fiMeta];  
FDStoredProc1.Command.FillParams(  
  FDStoredProc1.Params);
```

Parametry nie będą wypełniane

```
with FDStoredProc1.FetchOptions do  
  Items := Items - [fiMeta];
```

Ręczne wypełnianie w kodzie

```
FDStoredProc1.StoredProcName :=  
  'my_proc';  
with FDStoredProc1.FetchOptions do  
  Items := Items - [fiMeta];  
with FDStorecProc1.Params do begin  
  Clear;  
  with Add do begin  
    Name := 'Par1';  
    ParamType := ptInput;  
    DataType := ftString;  
    Size := 50;  
  end;  
  with Add do begin  
    Name := 'Par2';  
    ParamType := ptOutput;  
    DataType := ftInteger;  
  end;  
end;
```

Procedura z pakietu (Oracle)

```
FDStoredProc1.PackageName := 'SYS.DBMS_SQL';
FDStoredProc1.Overload := 1;
FDStoredProc1.StoredProcName := 'BIND_VARIABLE';
```

Procedura przeciążona

Wykonanie procedur składowanych

TFDStoredProc

```
with FDStoredProc1 do begin
  StoredProcName := 'MY_PROC';
  Prepare;
  Params[0].Value := 100;
  Params[1].Value := 'audi';
  ExecProc;
end;
```

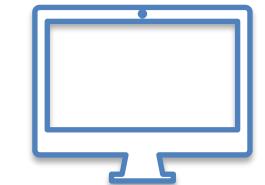
```
FDStoredProc1.ExecProc('MY_PROC',
  [100, 'audi']);
```

TFDQuery (składnia Oracle)

```
with FDQuery1 do begin
  SQL.Clear;
  SQL.Add('begin');
  SQL.Add('  sys.dbms_sql.
bind_variable(:c, :name, :value)');
  SQL.Add('end;');
  Params[0].AsInteger := 1;
  Params[1].AsString := 'p1';
  Params[2].AsInteger := 100;
  ExecSQL;
end;
```



FireDAC



Aktualizacja danych

Aktualizacja TFDTTable

Polecania DML

- INSERT, UPDATE, DELETE

Generowanie automatycznie

- Rozpoznawane pole auto-numerowane
- Rozpoznawane pola read-only

Proste i działa

Aktualizacja TFDQuery

Generowanie automatyczne

- Polecenia SQL DML są generowane automatycznie przez FireDAC
 - Na podstawie polecenia *SELECT*
 - Są budowane polecenia *INSERT, UPDATE i DELETE* (*nie tylko*)

Jedna tabela

- Polecenia aktualizują tylko jedną tabelę
- Jest to pierwsza tabela w poleceniu SELECT
- Można wskazać inną tabelę
 - FDQuery1.UpdateOptions.UpdateTableName

Podobnie jak w BDE

Trudności z aktualizacją TFDQuery

Rodzaj polecenia SQL	Trudności	Jak można sobie poradzić?
SELECT z JOIN-ami	Aktualizuje pola z dołączanych tabel	Usunięcie pól nie aktualizowanych
SELECT z GROUP BY	Aktualizuje więcej niż jeden rekord	Ustawienie w opcjach odpowiedniej tabeli
SELECT z nazwą tabeli wyliczaną wyrażeniem	Nie może rozpoznać nazwy tabeli	Ustawienie w opcjach odpowiedniej tabeli
SELECT z aliasami lub wyliczonymi kolumnami	Usunięcie pola, których nie ma w tabeli	Usunięcie pól nie aktualizowanych
Wywołanie procedury składowanej EXEC	Nie działa aktualizacja	
Wyrażenie z kursorem i zmienną	Nie działa aktualizacja	
Inne polecenia niż SELECT	Nie działa aktualizacja	

Wyłączanie aktualizacji dla pola

Jak usunąć zbędne pola z aktualizacji?

1) Nie branie pod uwagę w czasie generowania

- `TField.PropertyFlags` wyłącz `pfInUpdate`

```
Field1.ProviderFlags := Field1.ProviderFlags - [pfInUpdate];
```

2) Zablokowanie edycji pola

- `TField.ReadOnly`

```
Field1.ReadOnly := True;
```

Przemapowanie pola zaaliasowanego

Jeśli w polecenie SELECT jest pole nazwane aliasem

- Poinformowanie TField w jakim polu w bazie danych są zapisywane dane:
- TField.Origin

```
Field1.Origin := 'ORDERDATE';
```

Aktualizacja TFDStoredProc

Aktualizacja

- Przez TUpdateSQL
- Można wskazać tabelę
 - FDStoredProc1.UpdateOptions.UpdateTableName

TFDUpdateSQL

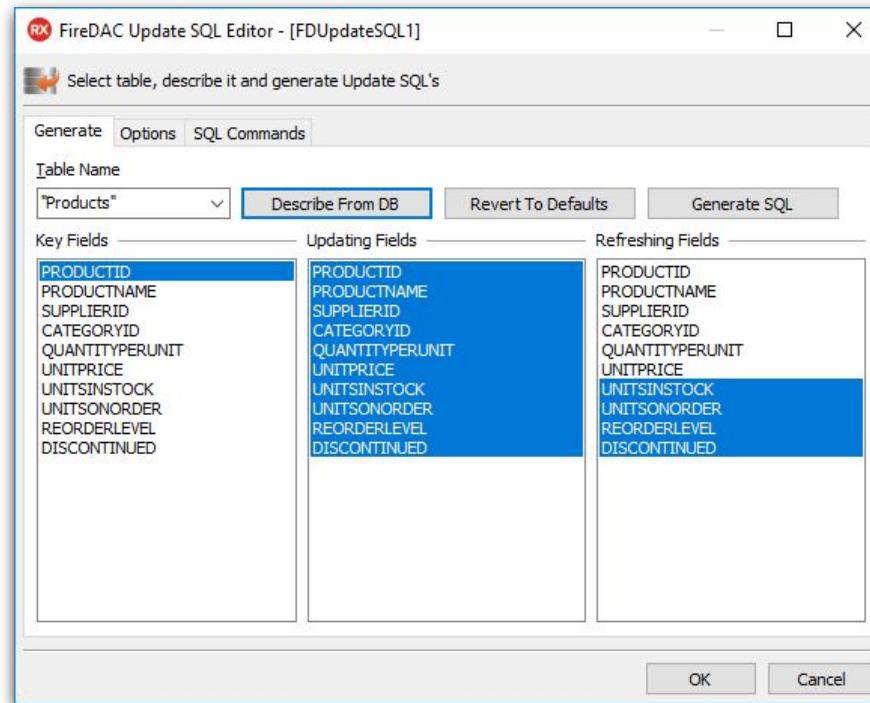
Połączenie z FDQuery

Połączenie z FDStoredProc

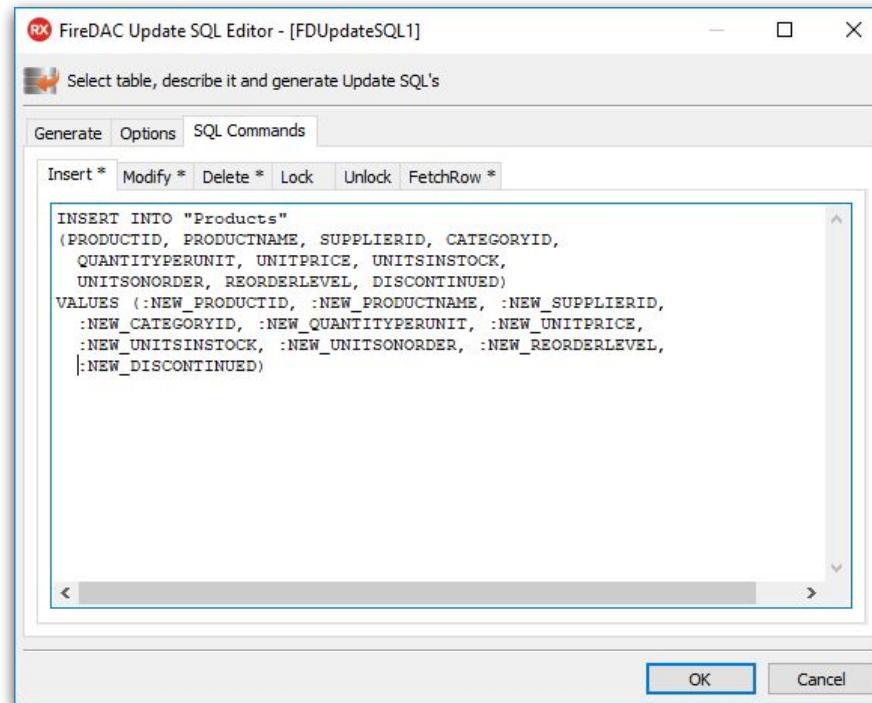
- FDQuery1.UpdateObject := FDUpdateSQL



UpdateSQL - kreator wizualny



UpdateSQL - kreator wizualny



UpdateSQL - generowanie z kodu

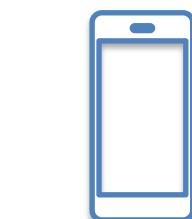
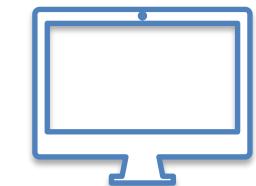
Lepsza kontrola niż wizualnie

- Własności (TStrings)
 - *ModifySQL, InsertSQL, DeleteSQL, FetchRowSQL*
 - (*opcja*) *LockSQL, UnlockSQL*

```
FDQuery1.UpdateObject := FDUpdateSQL1;
FDUpdateSQL1.ModifySQL.Text :=
  'UPDATE {id Products} SET '+
  ' ProductName = :NEW_ProductName, '+
  ' SupplierID = :NEW_SupplierID, CategoryID = :NEW_CategoryID, '+
  ' QuantityPerUnit = :NEW_QuantityPerUnit, UnitPrice = :NEW_UnitPrice, '+
  ' UnitsInStock = :NEW_UnitsInStock, UnitsOnOrder = :NEW_UnitsOnOrder, '+
  ' ReorderLevel = :NEW_ReorderLevel, Discontinued = :NEW_Discontinued '+
  ' WHERE ProductID = :OLD_ProductID ';
```



FireDAC



Widoki: sortowanie i filtrowanie wierszy

Sortowanie TFDQuery

```
FDQuery1.Open('select * from {id Customer} order by Country');
```

```
FDQuery1.IndexFieldNames := 'ORDERID';
```

```
FDQuery1.IndexFieldNames := 'OrderDate:D;Price';
```

```
with FDQuery1.Indexes.Add do begin
  Name := 'By OrderDate';
  Fields := 'OrderDate';
  Active := True;
end;
FDQuery1.IndexName := 'By OrderDate';
```

Filtrowanie TFDQuery

Filtrowanie wierszy

- TFDQuery.Filter
- TFDQuery.OnFilterRecords

Filtrowanie wg zakresu

- Zdefiniowanie zakresu
- Włączenie filtra

Filtrowanie wg statusu

Filtrowanie wierszy

```
FDQuery1.Filter := 'OrderID in (10150, 10151, 10152)';  
FDQuery1.Filtered := True;
```

```
FDQuery1.OnFilterRecord := Form1FilterRecord;  
FDQuery1.Filtered := True;  
  
procedure TForm1.Form1FilterRecord(DataSet: TDataSet;  
  var Accept: Boolean);  
var  
  iOrderID: Integer;  
begin  
  iOrderID := DataSet.FieldByName('OrderID').AsInteger;  
  Accept := (iOrderID = 10150) or (iOrderID = 10151) or  
    (iOrderID = 10152);  
end;
```

Filtrowanie wg zakresu

```
FDQuery1.IndexFieldNames := 'ORDERID;ORDERDATE';
FDQuery1.SetRangeStart;
FDQuery1.KeyExclusive := False;
FDQuery1.KeyFieldCount := 1;
FDQuery1.FieldByName('OrderID').AsInteger := 10150;
FDQuery1.SetRangeEnd;
FDQuery1.KeyExclusive := False;
FDQuery1.KeyFieldCount := 1;
FDQuery1.FieldByName('OrderID').AsInteger := 10152;
FDQuery1.ApplyRange;
```

Metody

- SetRangeStart
- EditRangeStart
- SetRangeEnd
- EditRangeEnd
- ApplyRange
- CancelRange
- SetRange

Właściwości

- IsRanged
- KeyExclusive
- KeyFieldCount

SetRange = RangeStart + SetRangeEnd + ApplyRange

Filtrowanie wg statusu

FDQuery1.FilterChanges

– Zbiór, lista elementów:

- *rtModified*
- *rtInserted*
- *rtDeleted*
- *rtUnmodified*
- *rtHasErrors*

Widok TFDQuery

Zdefiniowanie widoku

- Stworzenie indeksu
- Ustawienie
 - Selected := True
- Podstawienie nazwy indeksu
 - TFDQuery.IndexName

Widok pozwala

- Ustalić porządek
- Sortować wg wyrażenia
- Filtrować
- Tylko unikalne dane

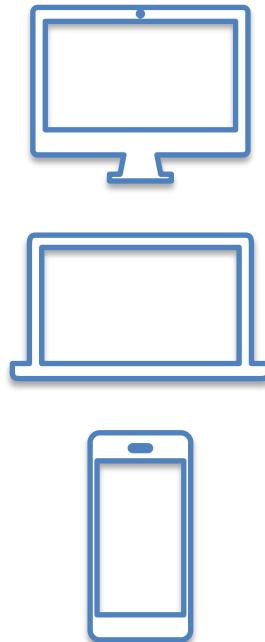
Nowy widok

- TFDIndex.Fields,
- TFDIndex.CaseInsFields
- TFDIndex.DescFields
- TFDIndex.Options
- TFDIndex.Expression
 - *Sortowanie wg wyrażenia*
- Filter
- FilterOptions
- TFDIndex.Distinct
 - *Powtarzające się wiersze są ignorowane*

```
with FDQuery1.Indexes.Add do begin
  Name := 'May sale dates';
  Fields := 'OrderDate';
  Filter := 'MONTH(OrderDate) = 5';
  Distinct := True;
  Active := True;
  Selected := True;
end;
```



FireDAC



Wyszukiwanie rekordów

Wyszukiwanie rekordów

Standartowe odszukiwanie

```
FDQuery1.IndexFieldNames := 'ORDERID';
isLocated := FDQuery1.Locate('ORDERID',
  10150, []);
if isLocated then
  ShowMessage('Order is found')
else
  ShowMessage('Order is not found');
```

Wykorzystanie „Filter”

```
FDQuery1.Filter :=
'Price >= 1000 and Price <= 2000';
if FDQuery1.FindFirst then
  ShowMessage('Found !');
...
if FDQuery1.FindNext then
  ShowMessage('Found !');
```

Rozszerzone odszukiwanie

```
isLocated := FDQuery1.LocateEx(
'Price >= 1000 and Price <= 2000',
[]);
```

Sekcja



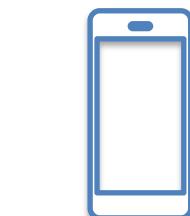
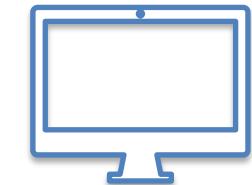
FireDAC

Zagadnienia zaawansowane

- Polecenia grupowe
- Master-detail
- Asynchroniczne wykonanie poleceń SQL
- LocalSQL
- Tryb offline oraz odzyskiwanie połączenia
- Tryb: Cached Updates



FireDAC



Polecenia grupowe

Command batches

Polecenia grupowe

Co to jest?

- Wysłanie do serwera równocześnie kilku poleceń SELECT

Dlaczego?

- Optymalizacja
- DBMS kompliluje polecenia grupy w jeden plan wykonania
- Polecenia w grupie są wykonywane na raz
- Zmniejszenie ruchu sieciowego i obciążenia serwera

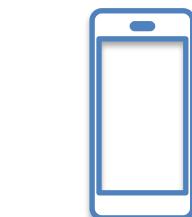
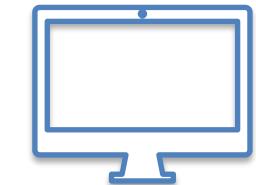
Polecenia grupowe

```
FDQuery1.FetchOptions.AutoClose := False;  
FDQuery1.SQL.Add('select * from [Orders];');  
FDQuery1.SQL.Add('select * from [Order Details]');  
FDQuery1.Open; // [Orders]  
FDQuery1.NextRecordSet; // [Order Details]
```

```
FDQuery1.FetchOptions.AutoClose := False;  
FDQuery1.SQL.Text := 'select * from orders; select * from customers';  
  
FDQuery1.Open;  
FDQuery1.FetchAll;  
// assign orders records to FDMemTable1  
FDMemTable1.Data := FDQuery1.Data;  
  
FDQuery1.NextRecordSet;  
FDQuery1.FetchAll;  
// assign customers records to FDMemTable2  
FDMemTable2.Data := FDQuery1.Data;
```



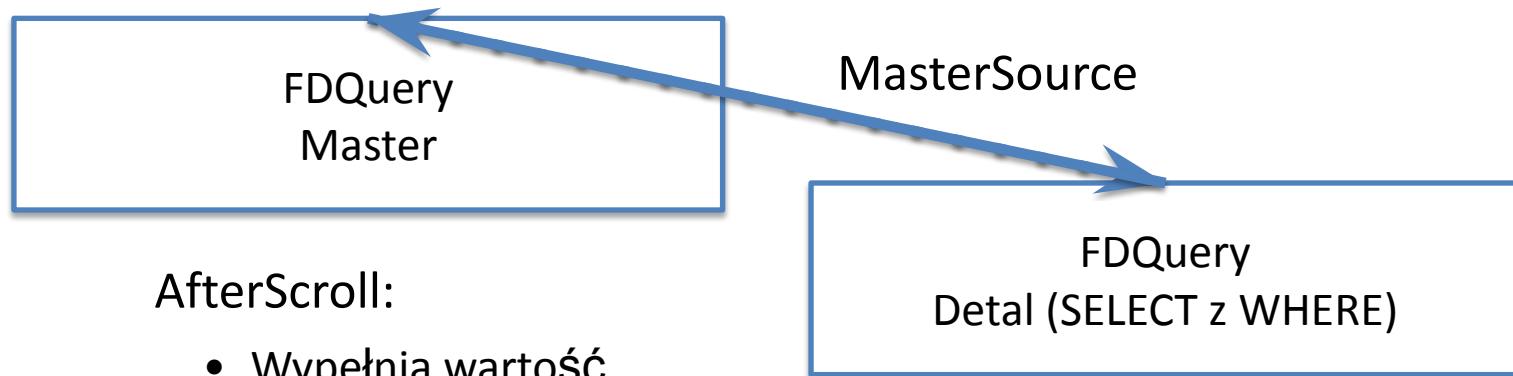
FireDAC



Master-detail

Relacja MD oparta o parametry

```
FDQuery1.Open('SELECT * FROM {id Orders}');
DataSource1.DataSet := FDQuery1;
FDQuery2.MasterSource := DataSource1;
FDQuery2.MasterFields := 'OrderID';
FDQuery2.SQL.Text :=
  'SELECT * FROM {id Order Details} WHERE OrderID = :OrderID';
FDQuery2.Open();
DataSource2.DataSet := FDQuery2;
```

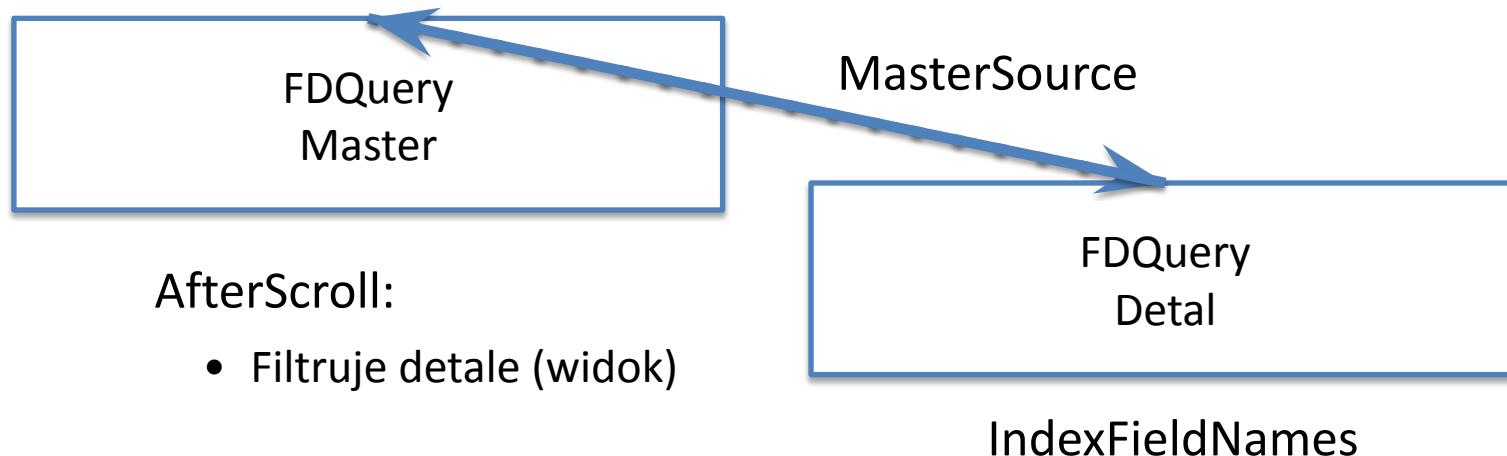


AfterScroll:

- Wypełnia wartość parametru w detalach
- Odświeża polecenie

Relacja MD oparta o zakres

```
FDQuery1.Open('SELECT * FROM {id Orders}');
DataSource1.DataSet := FDQuery1;
FDQuery2.MasterSource := DataSource1;
FDQuery2.MasterFields := 'OrderID';
FDQuery2.IndexFieldNames := 'OrderID';
FDQuery2.SQL.Text :=
  'SELECT * FROM {id Order Details}';
FDQuery2.Open();
DataSource2.DataSet := FDQuery2;
```



Master-detail - porównanie metod

Oparta o parametr

- ✓ Detale pobierają minimalną liczbę danych
- ✓ Detale są często odświeżane
- Przy nawigacji dużo zapytań trafia na serwer
 - optymalizacja: w detalach DetailDelay

Oparta o zakres

- ✓ Redukuje komunikację z serwerem
- ✓ Działa w trybie off-line
- ✓ Umożliwia skorzystanie z trybu Centralnego Buforowania Modyfikacji
- Jednorazowo pobiera dużo danych

Master-detail - przykłady

Kalkulacje na zbiorze master

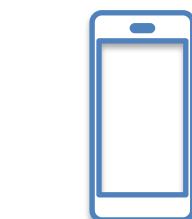
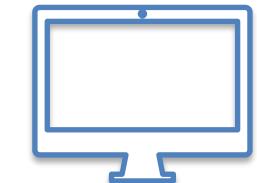
```
qOrders.DisableControls;  
try  
    qOrders.First;  
    while not qOrders.Eof do begin  
        .....  
        qOrders.Next;  
    end;  
finally  
    qOrders.EnableControls;  
end;
```

Manualne odświeżanie detali

```
qOrderDetails.MasterLink.DisableScroll;  
try  
    qOrders.First;  
    while not qOrders.Eof do begin  
        orderID := qOrders.  
            FieldByName('OrderID').AsInteger;  
        if orderID = 100 then begin  
            qOrderDetails.ApplyMaster;  
        end;  
        qOrders.Next;  
    end;  
finally  
  
    qOrderDetails.MasterLink.EnableScroll;  
end;
```



FireDAC



Asynchroniczne wykonanie poleceń SQL

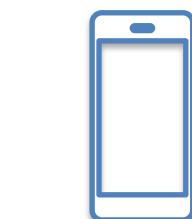
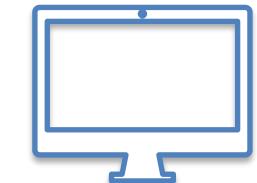
Asynchroniczne wykonanie poleceń

```
FDQuery1.ResourceOptions.CmdExecMode := amAsync;  
FDQuery1.Open;  
while FDQuery1.Command.State = csExecuting do begin  
    // do something while query is executing  
end;
```

```
procedure TForm1.FDQuery1BeforeOpen(DataSet: TDataSet);  
begin  
    DataSource1.DataSet := nil;  
end;  
  
procedure TForm1.FDQuery1AfterOpen(DataSet: TDataSet);  
begin  
    DataSource1.DataSet := FDQuery1;  
    FDQuery1.ResourceOptions.CmdExecMode := amBlocking;  
end;  
  
FDQuery1.BeforeOpen := FDQuery1BeforeOpen;  
FDQuery1.AfterOpen := FDQuery1AfterOpen;  
FDQuery1.ResourceOptions.CmdExecMode := amAsync;  
FDQuery1.Open;
```



FireDAC



LocalSQL

LocalSQL

Zapytania SQL na zbiorach TDataSet

- Każdy TDataSet zarejestrowany w TFDLocalSQL
- Silnik i składania SQLite



Zastosowania

- Zapytania łączące dane z różnych platform DB
- Baza danych w pamięci
- Zaawansowany tryb off-line
 - z możliwością dynamicznego generowania skomplikowanych zapytań
- Klient serwera aplikacyjnego
- Migracja pakietów DAC

LocalSQL przykład

```
FDLocalSQL1.Connection := FDConnection1;  
FDConnection1.DriverName := 'SQLite';  
FDLocalSQL1.Active := True;  
  
FDMemTable1.LocalSQL := FDLocalSQL1;  
with FDLocalSQL1.DataSets.Add do begin  
  DataSet := FDMemTable2;  
  Name := 'Models';  
end;  
  
FDQuery1.Connection := FDConnection1;  
FDQuery1.Open(  
  'select B.Name, M.Name Model, ModelYear, ' +  
  'Price from FDMemTable1 B left join'+  
  'Models M on B.Code = M.Code');
```

FDMemTable1

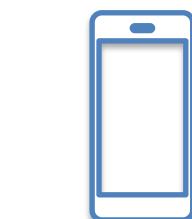
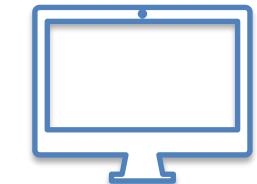
Code	Name
1	Audi
2	BMW
3	Mercedes

FDMemTable2

Code	Name	ModelYear	Price
1	A4	2012	40 000,00 zł
1	A6	2012	50 000,00 zł
1	Q7	2012	70 000,00 zł
2	3	2012	40 000,00 zł
2	5	2012	50 000,00 zł
2	X5	2012	70 000,00 zł
3	C	2012	40 000,00 zł
3	E	2012	50 000,00 zł
3	G	2012	70 000,00 zł



FireDAC



Pola auto-numerowane

Pola auto-numerowane

Pole typu Integer

- Kolumna z automatyczną inkrementacją

Warianty

- Identity
 - *Specjalnie stworzone pole IDENTITY / AUTOINC)*
- Sekwencja
 - *Modyfikowane przez wyzwalacze*
 - *Dodatkowo wymaga obiektu w bazie danych, który zawiera unikalną wartość. Sekwencja / Generator*
 - *Wymaga stworzenia sekwencji w bazie oraz dodania wyzwalaczy (przynajmniej BEFORE INSERT), które pobierają i wpisują wartość sekwencji do pola*

Obsługa w FireDAC

FireDAC rozpoznaje oba rodzaje pól auto-numerowanych

- Otrzymanie nowej wartości po stornie aplikacji klienckiej
- Obsługiwane w trybach:
 - *Natychmiastowych aktualizacji*
 - *Buforowanym (Cached Updates)*
- Rozpoznawane są wszystkie silniki baz danych
 - *Zobacz w dokumentacji temat: Auto-Incremental Fields (FireDAC)*

Pola tylu identity / autoinc

W czasie edycji

- Nadawane są numery: -1, -2 ...

W momencie zatwierdzenia

- Pobierana jest wartość pola auto-numerowanego

EmployeeID	LastName	FirstName	Title	Title
1	Davolio	Nancy	Sales Representative	M
2	Fuller	Andrew	Vice President, Sales	Dr
3	Leverling	Janet	Sales Representative	M
4	Peacock	Margaret	Sales Representative	M
5	Buchanan	Steven	Sales Manager	M
6	Suyama	Michael	Sales Representative	M
7	King	Robert	Sales Representative	M
8	Callahan	Laura	Inside Sales Coordinator	M
9	Dodsworth	Anne	Sales Representative	M
* -1	Bogdan	Polak		

Rozpoznawanie sekwencji

FireDAC stara się wykryć sekwencje

- Szuka dla tabeli BEFORE INSERT TRIGGER z sekwencją
- Sprawdza nazwę sekwencji

Uruchomienie generatora

- Natychmiastowe / Opóźnione
- TFDQuery.UpdateOptions.FetchGeneratorsPoint
- gpDeferred /gplImmediate

GetLastAutoGenValue

Funkcja

- TFDConnection.GetLastAutoGenValue
- Zwraca wartość sekwencji
- ostatnio wygenerowaną wartość sekwencji lub ostatnio nadaną wartość autoinc
- Parametr - wymagany w przypadku sekwencji
 - *Nazwa sekwencji lub generatora*

```
FDConnection1.GetLastAutoGenValue ('GEN_EMPLOYEE');

FDConnection1.GetLastAutoGenValue (''); // MS SQL, MySQL, itd.
```

Przykład

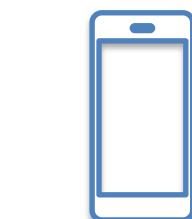
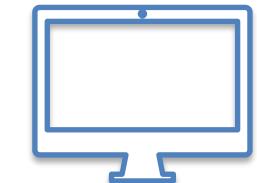
```
FDQuery1.SQL.Text := 'select * from {id Employees}';

FDQuery1.UpdateOptions.AutoIncFields := 'EMPLOYEEID';
FDQuery1.UpdateOptions.GeneratorName := 'GEN_EMPLOYEE';
FDQuery1.UpdateOptions.FetchGeneratorsPoint := gpImmediate;

FDQuery1.Open();
```



FireDAC



Buforowanie zmian

Tryb CachedUpdates

Inny tryb aktualizacji danych

- Polecenie „Post” nie wysyła modyfikacji do serwera

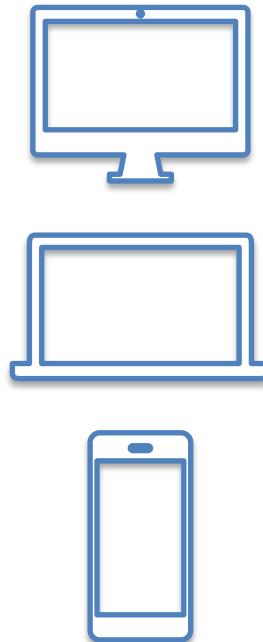
Pozwala pracować w trybie off-line

- Modyfikacje są zbierane w pamięci aplikacji (zbioru danych)
- Aktualizacja na serwerze następuje po wydaniu polecenia „ApplyUpdates”

Pozwala poprawić wydajność aplikacji



FireDAC



Centralized Cached Updates