# Využití nástrojů pro tvorbu databází v prostředí BlackBox Component Buildru

**Pavel Kořenek**

Katedra měřicí a řídicí techniky, VŠB-Technická univerzita Ostrava, tř. 17 listopadu, 708 33 Ostrava-Poruba, Česká republika

## *ÚVOD*

Databáze jsou v dnešní době běžně používány k řešení problémů spojených s uspořádáním dat do přehledné formy. Změna zpracovávaných dat však s sebou někdy přináší problém založení a správy nových dat-databáze. Myšlenka použití komponent pro často se opakující programové sekvence vede ke zjednodušení a zrychlení práce. Výběr vhodného nástroje pro implementaci pak může ušetřit dost času při vývoji aplikace. Podobných nástrojů je však na trhu velmi mnoho a lze je obecně rozdělit do několika kategorií - na vývojové nástroje založené na vlastním (propretárním) jazyce, nástroje založené na C++, nástroje pro Javu a nástroje pro webové aplikace. Zde jsou některé z nich: Delphi od fa Borland-Inprise, C++ Builder JBuilder, IntraBuilder, od Microsoft: Visual Basic, Visual C++, InterDev, Visual J++, od fa Sybase PowerBuilder, Power++, PowerJ, PowerSite, PowerDesigner, atd. Produkty každé firmy samozřejmě podporují své vlastní databáze, ale nabízena je i podpora databází konkurenčních. Je však velmi důležité že došlo jasně k přesunu a orientaci na vývojové nástroje typu RAD (Rapid Application Development). RAD je zkratka pro rychlý vývoj aplikací. V podstatě se jedná o rychlé prototypování vzhledu a částečně i funkcí aplikace.

### *BlackBox Component Builder*

Black Box Component Framework (BCB) je název prostředí ve kterém je implementován programovací jazyk Component Pascal. Je jedním z představitelů objektově a komponentně orientovaných nástrojů. Není to prostředí speciálně určené pro práci s databázemi, ale protože jde o poměrně nový produkt – vznikl ve firmě Oberon Microsystém, Inc. v roce 1992, přináší s sebou některé nové prvky a postupy, kterými se od ostatních liší.

BCB lze chápat jako produkt mající vlastnosti operačního systému i aplikace programovacího jazyku. Může být považován za run – time aplikaci tak jako operační systémy. Lze jej provozovat na všech běžně používaných operačních systémech. Je to integrovaný vývojový prvek, grafický návrhový prvek, prvek pro vytváření knihoven, textový procesor. BCB se dovede adaptovat k uživatelskému rozhraní, k platformě na které je právě spuštěn. Velký význam při tvorbě BCB byl kladen na podporu komponentního softwaru.

- BCB je schopen implementace na všechny standardní operační systémy jako jsou Windows, Mac, OS, OS/2, UNIX a je tedy na systému nezávislý.
- BCB programy jsou dále nezávislé na GUI použitého operačního systému
- BCB má architekturu dokumentů kompatibilní s OLE2 a OPENDOC

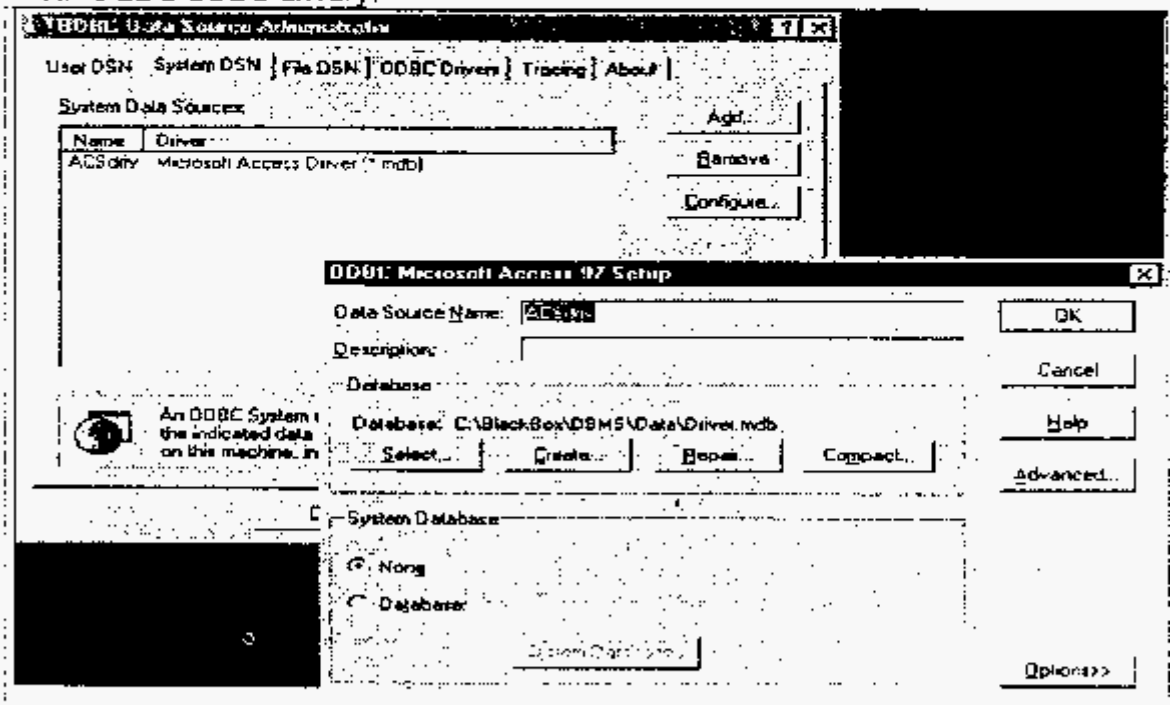
Je jednoduchý má možnost dynamického linkování má typovou kontrolu a je výkonný.

Ve spojení s jazykem Component Pascal (CP) se jedná o silný vývojový nástroj, protože Component Pascal zajišťuje lepší podporu komponentního frameworku.

Je podporováno OLE Automation. Je dostupná kompletní a typově bezpečná množina rozhraní na Microsoft Office97. To znamená, že mohou být například vytvářeny tabulky aplikace Excel přímo z prostředí BlackBox. Tato možnost je taktéž možná pouze z prostředí Windows. SQL subsystém je nyní standart. Je zde propojení na ODBC (takže lze používat ODBC ovladač a databáze) a na dtF SQL knihovny (single-user verze). Je zde možnost získání verze dtF z Internetu, pro nekomerční nebo zkušební potřeby. Nový komunikační subsystém poskytuje jednoduché rozhraní pro přístup ke komunikačním kanálům v plně duplexním režimu TCP/IP. Tato možnost je samozřejmě možná pouze z prostředí Windows.

### *DBMS*

Většina současných DBMS – databázových systémů (DataBase Management System) jsou založeny na spojení datového modelu a užití standardu SQL jazyka (Structured Query Language).
Subsystém BlackBoxu ***Sql*** zajišťuje jednoduchý, rozšiřitelný a integrovaný přístupový mechanismus k SQL databázím. Rozšiřitelný znamená – různé implementace databází mohou být přístupny díky vyhovujícímu driveru (modulu). Dnes existují tři základní nejpoužívanější drivery : pro dtF SQL knihovny, dtF klient/server knihovny a všeobecný Microsoft ODBC driver (Open DataBase Connectivity). BlackBox pracuje s dtF SQL a ODBC drivery.




Obrázek 1. : Nastavení ODBC driveru

### *Sql subsystém*

Slovo jednoduchost je klíčové v celém systému BCB. Je tedy příznačné, že i práce s databázemi v BCB za pomoci k tomu určenému subsystému ***Sql*** je snadná. V porovnání s jiným rozhraním na stejné úrovni flexibilty subsystém Sql poskytuje uživateli velmi snadno pochopitelné rozhraní. Centrální programové rozhraní je tvořeno modulem SqlDB, které poskytuje dva hlavní a několik minoritních datových

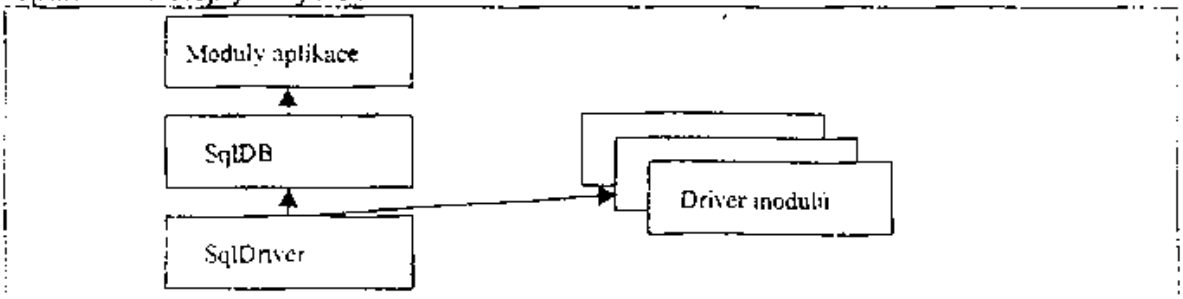
typů (Sql.Database, Sql.Table). Tím je vytvořena objektově orientovaná abstrakce pro libovolnou SQL databázi respektive tabulku.
Je však nutno říci, že subsystém Sql nelze chápat jako uzavřený a dokončený nástroj pro práci s databázemi. Počítá se s tím, že tento subsystém bude programátorem dále upravován a k tomu jsou v Sql subsystému moduly sloužící jako šablony-template

Integrace

SQL příkazy je většinou nutné tvořit za běhu aplikace a reagovat tím na právě aktuální stav systému. Tedy namísto formálních parametrů v příkazech SQL bývají vkládány parametry aktuální (názvy tabulek, sloupců). Subsystém Sql za použití nástrojů k tomu určených v BCB toto doplnění provádí automaticky. Tedy v praxi dynamicky interpretovaný druh složeného SQL příkazu je tvořen za pomoci názvů proměnných jazyka CP umístěných přímo v příkazu SQL.

Rozšiřitelnost

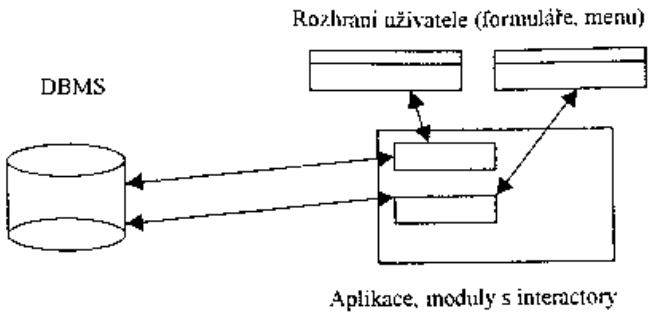
Programové aplikace používají rozhraní modulu SqlDB. Toto rozhraní je založeno na driveru modulu SqlDriver. Toto rozhraní driveru je vytvořeno v komponentně orientovaném stylu (je tím zajištěna jednoduchá rozšiřitelnost) a proto i použití různých driverů na jednu programovou aplikaci nebude mít chybný vliv na její správný chod. Vlastně lze použít několik různých konkurenčních driverů na stejnou aplikaci se stejným výsledkem.




**Obrázek 2. : Jednoduché rozhraní aplikací, rozšiřitelné rozhraní driverů**

Oddělené rozhraní GUI

Zkratka GUI - Grafical User Interface, je grafická část aplikace vytvořena programátorem v daném operačním sytému. V BCB je striktně oddělena platforma od rozhraní uživatele a jeho programové části. Grafické rozhraní vytvořené uživatelem je pouze kosmetickou úpravou programu a může být (vzhledem k striktnímu oddělení) kdykoli modifikováno bez zásahu do samotného kódu programu. Proto jakékoli příkazové tlačítka nebo vstupní textová pole použitá ve formuláři nemusí být součástí databáze nebo dokonce součástí kódu samotného. Program samotný pouze produkované a zpracovávané data skrze k tomu určené objekty – interactory (rekordy jazyka CP) spojuje. Proměnné takového typu, rekordu jsou používány jako zdrojové i cílové místa využívaná jak pro parametry příkazů tak i pro uložení výsledku samotného.




**Obrázek 3. : Interaktory jako spojnice mezi aplikací, DBMS a rozhraním uživatele**

***Databáze, tabulky, interactory***

Z pohledu programu je databáze representována SqlDB.Database objektem. SQL příkazy je možné provádět na databázi tak dlouho, dokud je objekt přístupný. Provedení příkazů může být charakteru lokálního nebo vzdáleného – na server.

SQL subsystém poskytuje dva hlavní typy SqlDB.Database a SqlDB.Table:

Zde je typ Database – bez atributů:

```
Database = POINTER TO ABSTRACT RECORD
        (d: Database) Exec (statement: ARRAY OF CHAR), NEW, ABSTRACT;
        (d: Database) Commit, NEW, ABSTRACT;
        (d: Database) Abort, NEW, ABSTRACT;
END;
```

Exec ... je používán pro vykonání SQL příkazu, který je odstartován příkazem Commit nebo zrušen pomocí Abort (můžeme mít připraveno v řadě za sebou několik příkazů SQL čekajících na vykonání, ale to je provedeno až příkazem Commit) a nemá návratovou hodnotu v podobě tabulky.

V případě, že se odkazujeme na databázi novou je třeba ji nejprve vytvořit (příkazem v typu SqlDB.Table) a následně otevřít – a k tomu slouží procedura

```
PROCEDURE OpenDatabase (protocol, id, password, datasource: ARRAY OF CHAR;
                        async: BOOLEAN; OUT d: SqlDB.Database;OUT res: INTEGER);
```

Zde specifikujeme protokol databáze a datový zdroj. Tedy protokol je „SqlOdbc" a datový zdroj je nastaven v 32ODBC driveru nastaven na cestu, kde je očekávána samotná tabulka databáze. Můžeme zde ještě říci, zdali budeme s databází pracovat v synchronním nebo asynchronním módu, určit si klíč databáze, přidat heslo.

Zde je typ Tabulka – bez atributů :

```
Table = POINTER TO ABSTRACT RECORD
        (t: Table) Exec (statement: ARRAY OF CHAR), NEW, ABSTRACT;
        (t: Table) Read (row: INTEGER; VAR data: ANYREC), NEW, ABSTRACT;
        (t: Table) Clear, NEW, ABSTRACT;
        (t: Table) Call (command: TableCommand; par: ANYPTR), NEW, ABSTRACT;
```

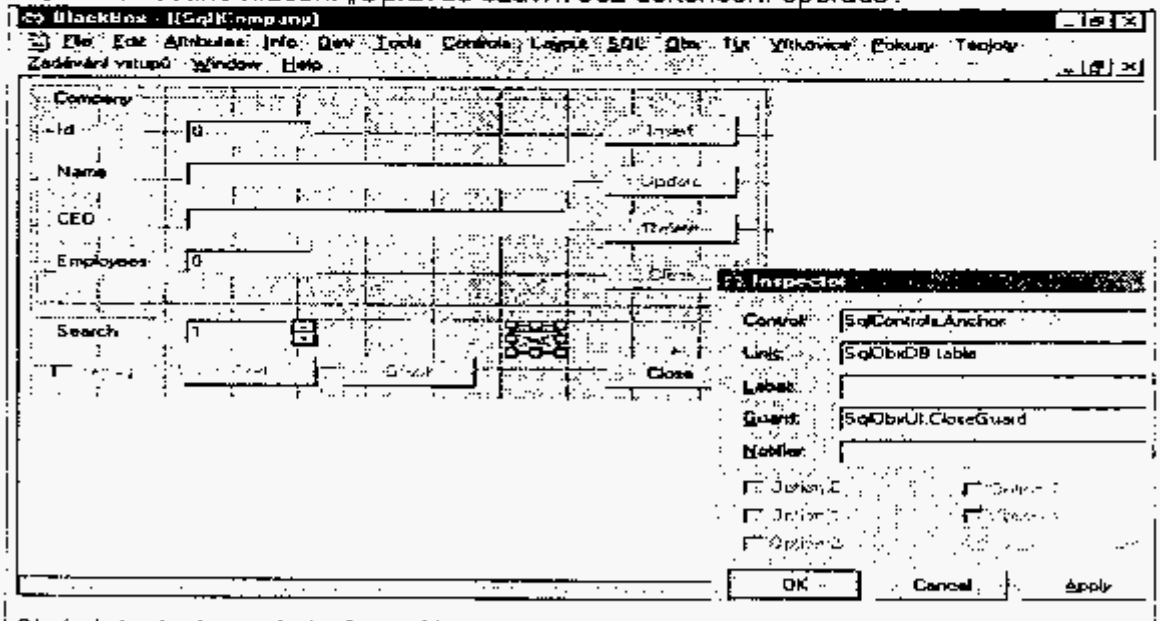
Exec ... je zde ve stejném významu jako u typu databáze, ale vrací hodnotu ve formě tabulky „table.Exec("SELECT * FROM Companies WHERE id = 17)"

Jestliže se vrátíme k integraci, pak využití proměnných v SQL příkazu by mohlo vypadat následovně:

```
ConvertIntegerToString(searchId, str);                  (*konvertuji číslo na string*)
str := "SELECT * FROM Companies WHERE id = " + str;     (*příkaz ve formě stringu *)
table.Exec(str);                        (*využití stringu – proměnné jako SQL příkazu *)
```

Uvolňování paměti

BCB obsahuje velmi efektivní nástroj, použitelný nejenom při práci s databázemi. Jedná se o tzv. automatický úklid paměti – garbage colector. Prostředí jazyka C++ zajišťuje explicitně pointry, ale nezajišťuje správu paměti. V prostředí JAVA je zase zajištěna správa paměti, ale chybí explicitné pointry. Component Pascal zajišťuje oboje, jak použití pointrů, tak i automatickou správu paměti. Jsou povoleny pouze typově bezpečné operace s pointry (ne aritmetické operace) a všechny pointry jsou automaticky inicializovány na hodnotu NIL. V CP nenalezneme žádné operace pro mazání paměti nebo operace dispose pro uvolnění paměti. O vše se stará automatický úklid paměti. V našem případě pro uvolnění alokované paměti pointry ukazujícími v rámci databáze např. na SQL příkaz, je nutné použít k tomu určený kontrolní prvek tzv. anchor controler. Ten je uložen v rámci subsystému Sql v modulu SqlControls. Anchor je pak spojen s globální proměnnou typu Table a zajišťuje, že po dobu inicializace příkazů, vykonávání SQL příkazů, po dobu otevření dialogových oken a práce s tabulkou, pointry ukazují na paměťová místa. Při ukončení práce s tabulkou (např. zavření celého databázového formuláře) controler zajistí uvedení všech používaných pointrů na hodnotu NIL. Toto je zvláště výhodné ve spojení s procedurou „Guard", která je definována uživatelem. Tuto proceduru je možné spojit s controlerem Anchor. V tomto případě by jejich spojení bylo velmi užitečné v případě, že uživatel chce ukončit práci s databází (uzavření formuláře) a Anchor zjistil, že např. některé vstupní textové pole formuláře je vyplněno textem, ale uživatel neprovedl operaci – třeba vložení (insert) údajů do tabulky. Zde je vyvolána uživatelem definovaná procedura „Guard", která vyvolá na monitor některé z předem BCB definované hlášení „Opravdu uzavřít bez dokončení operace?"

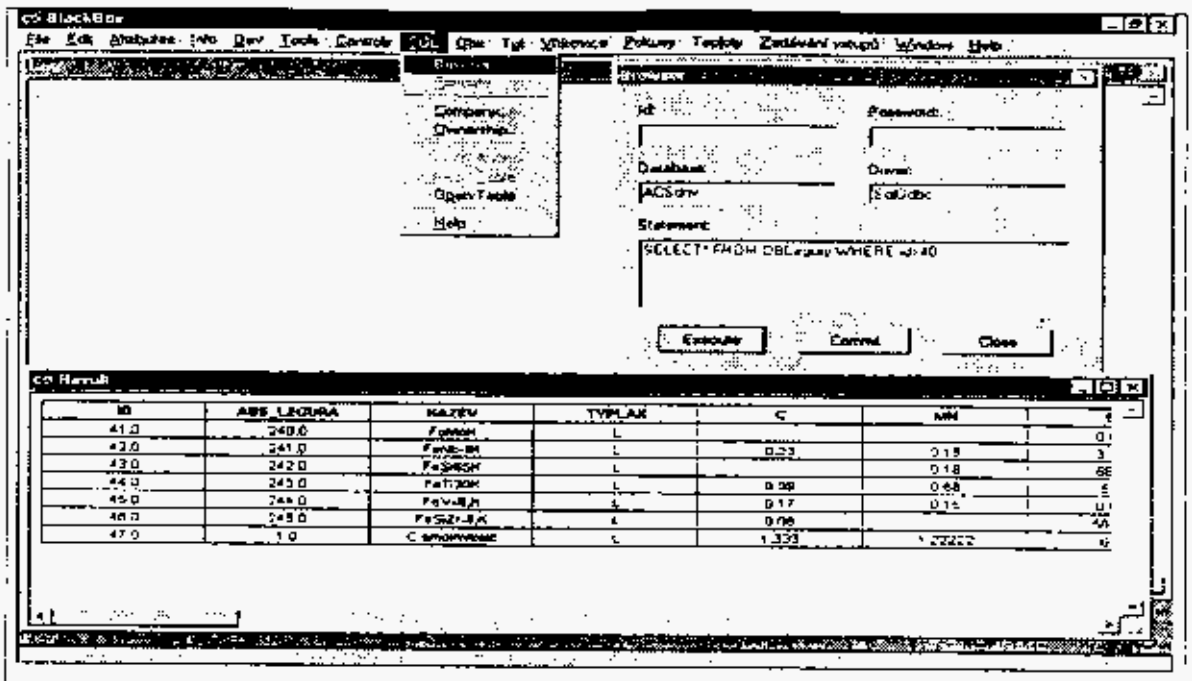



**Obrázek 4. : Anchor v návrhu formuláře s napojením na proceduru typu „Guard"**

Zobrazení tabulky

Často je nezbytné výsledek provedené operace SQL příkazu zobrazit. Zobrazená tabulka needitovatelná. Její zobrazení je možné vyvolat již BCB vytvořenou komponentou ze subsystému SqlControls. Zobrazení tabulky je možné také vyvolat pomocí procedury „Notifier", spojené s některým z tlačítek formuláře.

„TableNotifier=PROCEDURE (t:SqlDB.Table;row,column:INTEGER.modifers:SET)"




Obrázek 5. : Výsledek SQL příkazu v podobě tabulky

```
MODULE SqlBrowser;
IMPORT Dialog, Views, TextModels, TextControllers, SqlDB, SqlControls;
VAR    dlg*: RECORD
id*,      password*,      database*,      driver*: ARRAY 32 OF CHAR;      statement*: ARRAY 1024 OF
CHAR
        END;
table*: SqlDB.Table;      (* anchor for database *)

PROCEDURE CheckResult (tab: SqlDB.Table; par: ANYPTR);
VAR v: Views.View;
        BEGIN
        IF tab.res = 0 THEN
                IF tab.columns > 0 THEN      v := SqlControls.dir.NewTableOn(tab);   Views.OpenAux(v, "#Sql:Result")
                ELSE Dialog.ShowMsg("#Sql:StatementExecuted")
                END
        ELSE Dialog.ShowMsg("#Sql:ExecutionFailed")
        END
END CheckResult;

PROCEDURE ExecuteThis (statement: ARRAY OF CHAR);
VAR res: INTEGER; db: SqlDB.Database; tab: SqlDB.Table;
        BEGIN
        IF table = NIL THEN
SqlDB.OpenDatabase(dlg.driver, dlg.id, dlg.password, dlg.database,SqlDB.async, SqlDB.showErrors, db, res);
                IF res = 0 THEN              table := db.NewTable()
                ELSIF res <= 3 THEN          Dialog.ShowMsg("#Sql:CannotLoadDriver")
                ELSE                         Dialog.ShowMsg("#Sql:ConnectionFailed")
                END
        END;
        IF (statement # "") & (table # NIL) THEN
                (* allocate separate table to allow for multiple open tables *)
                tab := table.base.NewTable();          tab.Exec(statement);
                (* separate result check from execution to allow for asynchronous operation *)
                        tab.Call(CheckResult, NIL)
                END
END ExecuteThis;

PROCEDURE Execute*;
BEGIN           ExecuteThis(dlg.statement)
END Execute;

PROCEDURE ExecuteSel*;
VAR c: TextControllers.Controller; r: TextModels.Reader; beg, end, i: INTEGER; stat: ARRAY 1024 OF CHAR;
```

```
BEGIN
        c := TextControllers.Focus();
          IF (c # NIL) & c.HasSelection() THEN    c.GetSelection(beg, end);      r := c.text.NewReader(NIL);
          r.SetPos(beg), i := 0;                  WHILE r.Pos() < end DO r.ReadChar(stat[i]); INC(i) END.
          stat[i] = 0X;                           ExecuteThisstat()
          END
END ExecuteSel.

PROCEDURE Commit*;
BEGIN
        IF table # NIL THEN          table base.Commit          END
END Commit;

BEGIN
        dlg.id := ""; dlg.password := ""; dlg.database := "ACSdriv"; dlg.driver := "SqlOdbc"
END SqlBrowser.
```

**Obrázek 6. : Kód modulu BROWSER: provede SQL příkaz na libovolné tabulce s definovaným driverem v SqlOdbc formátu s následným zobrazení výsledku SQL příkazu v tabulce.**

## Závěr

BlackBox Component Pascal nevznikl za účelem nahradit současné databázově orientované vývojové prostředí. Ale svým přístupem se kterým přistupuje k řešení všech úloh (metoda odděleného návrhu modelů, formulářů, pohledů a interactorů-spojovacího článku), se jeví ve spojení s jazykem Component Pascal jako velmi silný vývojový nástroj – jednoduchý a snadný. Architektura BCF přináší velké množství nových vzorů a postupů. Některé z nich jsou specifické pouze pro BlackBox: carrier-rider-mapper separace (CRM separace), directory objekty, HMVC, ke kterým se vyjadřují jiné příspěvky v tomto sborníku autorů M. Brožka, J. Černohorského.


[1] SZYPERSKI, C. Component software: beyond object-oriented programming. 1.vyd. Harlow: Addison-Wesley, 1997.411 s.

[2] The Oberon System, User guide and Programmer´s manual, Martin Reiser

[3] Warford, J., Programming in BlackBox –The PBox Project. ftp://ftp.pepperdine.edu/pub/compsci/prog-bbox